

J. McAllister

**Artificial
Intelligence
and PROLOG
on Microcomputers**



Дж. Макаллистер

**Искусственный
интеллект
и Пролог
на микроЭВМ**

Дж. Макаллистер

**Искусственный
интеллект
и Пролог
на микроЭВМ**

Artificial Intelligence and PROLOG on Microcomputers

J. McAllister

Senior Lecturer in Electronics and Mathematics
Kingston College of Further Education



Edward Arnold

Дж. Макаллистер

Искусственный интеллект и Пролог на микроЭВМ

Перевод с английского
А. В. Чукашова, М. В. Сергиевского

Под редакцией
канд. техн. наук М. В. Сергиевского



Москва
•Машиностроение•
1990

ББК 32.973.2-01

М15

УДК [007 : 159.955] : 519.768.2=20

Макаллистер Дж.

- М15** Искусственный интеллект и Пролог на микроЭВМ/Пер с англ. А. В. Чукашова, М. В. Сергиевского; Под ред. М. В. Сергиевского.— М.: Машиностроение, 1990.— 240 с.: ил.
ISBN 5-217-00973-X

В книге английского автора рассматривается использование наиболее популярного языка логического программирования для построения баз данных, баз знаний и экспертных систем. Книга содержит сравнительно мало теоретического материала и ориентирована в основном на практическое использование Пролога. Достаточно подробно описаны возможности этого языка.

Для программистов и разработчиков систем искусственного интеллекта. Может быть использована как учебное пособие на курсах повышения квалификации соответствующих специалистов.

М $\frac{1402070000-134}{038 (01)-90}$ 134—90

ББК 32.973.2-01

ISBN 5-217-00973-X (СССР) © J. McAllister, 1987

ISBN 0-7131-3611-1 (Велико-
британия) © Перевод на русский язык А. В. Чукашова, М. В. Сергиевского и предисловие редактора перевода М. В. Сергиевского, 1990

ОГЛАВЛЕНИЕ

Предисловие редактора перевода	7
Предисловие автора	8
1. Искусственный интеллект	10
1.1. История развития искусственного интеллекта	10
1.2. Основные задачи искусственного интеллекта	13
1.3. Наиболее известные системы искусственного интеллекта	16
1.4. Инженерия знаний	20
1.5. Другие экспертные системы	22
1.6. Японский проект создания ЭВМ пятого поколения	25
1.7. Замечания для начинающих	26
2. Введение в Пролог	28
2.1. Почему следует изучать Пролог?	28
2.2. Основные понятия	29
2.3. Отношения	30
2.4. Синтаксис предложений Пролога	34
2.5. Модульное программирование	37
2.6. Информация для пользователей ЭВМ Spectrum	37
2.7. Запросы	38
2.8. Средства, позволяющие работать с числовой информацией	42
2.9. Отношения, задаваемые пользователем	45
2.10. Отношения, служащие для ввода данных	48
2.11. На пути к естественному языку	55
<i>Ответы к упражнениям</i>	<i>64</i>
3. Списки и базы данных	67
3.1. Список как структура данных	67
3.2. Использование списков в базах данных	70
3.3. Списки, состоящие из списков	73
3.4. Правила или знания	80
3.5. Модификация программы	84
3.6. Увеличение скорости и эффективности программ	90
3.7. Рекурсивные определения	102
3.8. Хвостовая рекурсия и списки	106
<i>Ответы к упражнениям</i>	<i>111</i>
4. Логика	114
4.1. Комбинации высказываний	115
4.2. Булева алгебра	116
4.3. Логическое следование (импликация)	118
4.4. Логические символы	119
4.5. Логические функции от двух переменных	120
4.6. Использование Пролога для доказательства теорем	122
4.7. Получение таблиц истинности с помощью Пролога	128
4.8. Правила де Моргана	132
4.9. Использование Пролога для реализации правил де Моргана	133

4.10. Логическое следование в Прологе	136
4.11. Пролог и цифровые логические устройства	144
4.12. Использование логических элементов для доказательства теорем	147
<i>Ответы к упражнениям</i>	150
5. Базы знаний	152
5.1. Реляционная база знаний	153
5.2. Модификация записей	160
5.3. Средства модификации записей	163
5.4. Миннаторная СУБД	168
5.5. Инициализация базы данных	169
5.6. Обновление базы данных	172
5.7. Использование базы данных	175
<i>Ответы к упражнениям</i>	189
6. Экспертные системы	193
6.1. Специализированная система	202
6.2. Способность к обучению	207
6.3. Использование экспертных систем для управления процес- сами	210
6.4. Использование экспертных систем при принятии решений . .	216
6.5. Системы, объясняющие логику своей работы	222
6.6. Система, способная к обучению	225
<i>Ответы к упражнениям</i>	234
Список литературы	237

Рассматриваемая версия языка Пролог ориентирована на бытовую ЭВМ фирмы Spectrum. Основным отличием бытовых ЭВМ от персональных является малая оперативная память (часто 64К). С этим связаны разного рода неприятности, борьбе с которыми в книге уделяется некоторое внимание. Поскольку у нас подобных ЭВМ пока нет, а доступные персональные ЭВМ имеют значительно большую память, у читателей, имеющих доступ к машине, таких проблем не возникает. В целом описываемая версия языка мало чем отличается от широко распространенных версий Пролога для микроЭВМ, работающих под управлением операционных систем CP/M, MS DOS и UNIX.

Несколько слов о содержании книги. Непосредственно вопросам искусственного интеллекта посвящена только первая глава. Остальные пять глав целиком связаны с Прологом и различными аспектами его использования. Анализируются возможности Пролога по обработке списков, работе с базами данных и знаний, доказательству теорем и построению экспертных систем.

Если считать, что все только что перечисленное составляет предмет исследований искусственного интеллекта, то можно сказать, что мы имеем дело с использованием Пролога для решения задач искусственного интеллекта. Но все же речь в книге идет, в первую очередь, о методах и приемах программирования на Прологе. Показывается, как эти методы зависят от типа предметной области: для области доказательства теорем они одни, для проектирования экспертных систем — другие. В этом отношении книга дополняет и развивает материал ранее выпущенных в нашей стране книг У. Клоксина и К. Кларка.

Книга отличается большим числом примеров программ и подробным описанием их работы. Эти примеры представляют собой образцы эффективного стиля программирования, опирающегося на широкое использование списковых структур и других встроенных средств языка. В связи с этим при подготовке книги к изданию структура текстов программ была сохранена, только кое-где были добавлены комментарии на русском языке. Имена отношений решено было не переводить, но в большинстве протоколов работы программ для каждого сообщения ЭВМ в квадратных скобках дан его перевод.

Предисловие автора, гл. 1, 4 и 5 переведены А. В. Чукашовым, гл. 2, 3 и 6 — М. В. Сергеевским.

М Сергеевский

Публикации, посвященные искусственному интеллекту, можно условно разделить на два вида. К первому из них относятся книги по теории искусственного интеллекта, написанные специалистами в этой области, как правило, с целью, далекой от практического применения. Такие публикации предназначены для квалифицированных программистов, имеющих опыт работы с системами искусственного интеллекта, и служат для углубления знаний. Большая часть пользователей микроЭВМ вряд ли сможет на практике воспользоваться описываемыми в этих книгах программами и системами. Публикации второго вида, наоборот, ориентированы на массового пользователя микроЭВМ и предназначены прежде всего для практического применения. Фрагменты программ, иллюстрирующие рассматриваемые в них алгоритмы, представляются обычно на языке Бейсик.

Автор предлагаемой книги старался придерживаться некоторого среднего подхода между этими двумя крайностями. Теоретический материал, излагаемый здесь, несложен и поясняется примерами программ на языке Пролог. Этот язык программирования распространен в настоящее время в основном в Европе. Однако популярность его постоянно растет. Он выбран японскими специалистами как один из основных языков для создания программного и аппаратного обеспечения ЭВМ пятого поколения. В США этот язык находят все большее применение для ЭВМ с аппаратной реализацией языка Лисп, служащего для решения задач искусственного интеллекта и, кроме того, используемого как базовый при разработке трансляторов с Пролога. В последнее время Пролог получает все большее распространение на микроЭВМ. С появлением ряда мощных ЭВМ этого класса, снабженных накопителями на магнитных дисках большой емкости, стала возможной реализация небольших, но полезных на практике систем. Язык Лисп в настоящее время также используется на ряде микроЭВМ (например, на модели Sinclair QL). Однако для эффективного применения этого языка необходима специальная среда, поддерживаемая аппаратурой. Поэтому решение реальных задач с его помощью пока затруднено.

В этой книге главное внимание уделяется решению небольшого числа основных задач искусственного интеллекта. Однако

по приводным фрагментам программ можно получить представление о принципах создания сложных систем искусственного интеллекта. Автор надеется, что материал книги поможет читателю избежать составления длинных, узко специализированных программ на языке Бейсик и перейти к значительно более коротким универсальным программам на Прологе.

Книга рассчитана в первую очередь на самостоятельную работу, и все ее главы, за исключением первой, содержат упражнения с ответами. Однако ее материал может быть использован и в качестве основы для соответствующих учебных курсов.

Автор не стремился рассматривать теоретически сложные задачи искусственного интеллекта, такие, например, как распознавание речи человека. Основное внимание уделялось проблемам, которые пользователи микроЭВМ могут решить, не углубляясь в сложности теоретических построений.

Дж. Макаллистер, 1987 г.

1. ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

1.1. ИСТОРИЯ РАЗВИТИЯ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

В последнее время в самых различных публикациях по вычислительной технике все чаще встречаются термины: «ЭВМ пятого поколения», «искусственный интеллект», «экспертные системы», а также названия пока мало распространенных в кругу программистов языков Лисп и Пролог. Раньше с понятием искусственного интеллекта (ИИ) связывали надежды на создание мыслящей машины, способной соперничать с человеческим мозгом и, возможно, превзойти его. Эти надежды, на долгое время захватившие воображение многих энтузиастов, так и остались несбывшимися. И хотя фантастические литературные прообразы «умных машин» создавались еще за сотни лет до наших дней, лишь с середины тридцатых годов, с момента публикации работ А. Тьюринга, в которых обсуждалась реальность создания таких устройств, к проблеме ИИ стали относиться серьезно. Для того чтобы ответить на вопрос, какую машину считать «думающей», Тьюринг предложил использовать следующий тест: испытатель через посредника общается с невидимым для него собеседником — человеком или машиной. «Интеллектуальной» может считаться та машина, которую испытатель в процессе такого общения не сможет отличить от человека.

Если испытатель при проверке компьютера на «интеллектуальность» будет придерживаться достаточно жестких ограничений в выборе темы и формы диалога, этот тест выдержит даже самый маломощный современный бытовой компьютер. Можно было бы считать признаком интеллектуальности умение поддерживать беседу, но, как было показано, эта человеческая способность легко моделируется на ЭВМ.

Признаком интеллектуальности может служить способность к обучению. В 1961 г. профессор Д. Мичи, один из ведущих английских специалистов по ИИ, описал механизм, состоящий из 300 спичечных коробков, который мог «научиться» играть в «крестики и нолики». Мичи назвал это устройство MENACE (Matchbox Educable Naughts and Crosses Engine). В названии («менасе» в переводе на русский означает «угроза») заключается, очевидно, доля иронии, вызванной предубеждениями перед «думающими машинами».

До настоящего времени единого и признанного всеми определения ИИ не существует, и это не удивительно. Достаточно вспомнить, что универсального определения человеческого интеллекта также нет. Дискуссии о том, что можно считать признаком ИИ, а что — нет, напоминают споры средневековых философов, которых интересовало, сколько ангелов смогут разместиться на кончике иглы. Сейчас к ИИ принято относить ряд алгоритмов и программных систем, отличительным свойством которых является то, что они могут решать некоторые задачи так, как это делал бы размышляющий над их решением человек. В предлагаемой книге мы не будем касаться вопросов, связанных с программированием сложных вычислительных задач. Нас будут больше интересовать процедуры решения, в которых используются такие свойства человеческого разума, как способность к абстрагированию и обобщению, как умение изобретать, обучаться и запоминать. Мы не собираемся искать определение ИИ. Вместо этого в книге рассматриваются основные алгоритмы, применяемые в настоящее время в системах ИИ.

Термин «пятое поколение» используется в японском долгосрочном проекте построения ЭВМ совершенно нового типа. В этих вычислительных машинах, кроме использования новой элементной базы и архитектуры, планируется применять нетрадиционный подход к решению задач с помощью новых языков программирования. Чтобы показать, почему именно сейчас понадобились ЭВМ нового типа, сделаем небольшой обзор истории развития аппаратного и программного обеспечения, начиная со времени появления первых вычислительных машин и до наших дней.

При изготовлении ЭВМ первого поколения использовались электровакуумные лампы. Эти компьютеры появились в 50-х годах и были созданы на основе специальных вычислительных устройств, использовавшихся во второй мировой войне в армиях Англии и США. Основное назначение этих устройств заключалось в расшифровке закодированных сообщений противника. ЭВМ этого поколения занимали очень много места — их часто приходилось размещать в специальных зданиях. Разработка, монтаж и эксплуатация таких машин требовали больших затрат. Кроме того, ЭВМ первого поколения потребляли огромное количество электроэнергии, и поэтому возникало множество проблем с охлаждением их отдельных узлов. По современным стандартам эти машины обладали довольно низкой производительностью, т. е. малым быстродействием, и к тому же были ненадежны. Программирование для таких компьютеров осуществлялось на самом низком уровне — в кодах, а подготовка программистов для каждой модели ЭВМ требовала большого труда.

Примерно в конце 50-х годов появились ЭВМ второго поколения, построенные преимущественно на полупроводниковых приборах. Эти машины были уже меньше по размерам, но для их работы требовалось все еще довольно большое количество элект-

троэнергии и во время эксплуатации они выделяли много тепла. Появление более производительных и сложных ЭВМ стало возможным во многом благодаря развитию технологии печатных схем, обеспечившему массовое производство сложных электронных блоков, которые в случае их порчи могли легко заменяться на исправные. В это же время для программирования начинают применять языки высокого уровня — ФОРТРАН, АЛГОЛ и КОБОЛ. Эти языки были еще довольно сложны для начинающих программистов, но для их использования уже не нужны детальные знания об устройстве и функционировании ЭВМ. Для начинающих программистов был создан язык Бейсик, и многие из программистов тех лет знакомились со своей профессией с его помощью.

ЭВМ третьего поколения стали создаваться в 60-х годах. В этих машинах использовались уже интегральные схемы (ИС). По внешнему виду, а также по функциональным принципам такие ЭВМ похожи на современные. Возросла производительность вычислительных машин — возросла и потребность в памяти, которая, в основном, была удовлетворена, когда появились быстродействующие закладываемые устройства на полупроводниках, заменившие устаревшие, громоздкие и низкоскоростные устройства на магнитных сердечниках. Плотность размещения элементов на ИС первых выпусков была невелика — не более десяти на каждой ИС. Однако вскоре начали выпускать так называемые СИС и БИС — интегральные схемы со средним и большим уровнями интеграции. На одной СИС могли разместиться до 100 элементов, тогда как на БИС их число достигало 1000. Размеры ЭВМ заметно уменьшились, а надежность, быстродействие и разрядность процесса увеличились. Производство ЭВМ стало автоматизированным, поэтому их стоимость упала настолько, что даже небольшие фирмы смогли заняться проектированием и выпуском собственных машин.

Для ЭВМ четвертого поколения понадобились уже сверхбольшие ИС (СБИС) с плотностью размещения, превышающей 10 000 элементов на схему. Появление СБИС дало импульс к развитию сразу нескольких направлений вычислительной техники. Число команд процессоров современных ЭВМ увеличилось, а их команды стали разнообразнее. Одновременно появились ЭВМ, имеющие процессоры с упрощенной архитектурой и с уменьшенным набором команд. Это так называемые РИСК-процессоры (RISC — Reduced Instruction Set Computer).

ЭВМ последнего типа играют сейчас важную роль в развитии вычислительной техники. Большие объемы недорогой оперативной памяти ЭВМ четвертого поколения позволили более эффективно использовать языки высокого уровня, а также устанавливать на указанных машинах мощные операционные системы. Функции управления и обработки, прежде выполнявшиеся различными блоками ЭВМ, в настоящее время реализуются одним устройством — центральным процессором (ЦП), что позволило обойтись

без теперь уже лишних транзисторов и ИС более низких уровней интеграции, которые использовались в машинах предыдущих поколений. Появились комплекты функциональных блоков ЭВМ на интегральных схемах, позволяющие создавать различные варианты микро- и мини-ЭВМ.

Вычислительная техника перестала быть замкнутой областью, доступной лишь специалистам, — компьютеры вошли почти во все сферы жизни человека. В создании ЭВМ четвертого поколения большую роль сыграло развитие МОП-технологии (технологии производства полупроводниковых приборов типа металл — окисел — полупроводник), позволяющей создавать электронные устройства с элементами, практически невидимыми без микроскопа. Устройства, созданные по такой технологии, для своей работы требуют ничтожных энергетических затрат. В этой книге технические детали построения ЭВМ не обсуждаются, но следует помнить следующий принцип: чем меньше по размерам электронное устройство, тем оно может работать быстрее. Скорость работы сейчас особенно важна, поскольку очень часто для решения задач на ЭВМ требуется обрабатывать огромные совокупности данных в режиме реального времени, когда каждая отдельная операция должна выполняться с максимальным быстродействием.

Многие специалисты в настоящее время придерживаются мнения, что для дальнейшего развития вычислительной техники нужны радикальные изменения архитектуры ЭВМ, так как для машин с обычной архитектурой предел возможностей достигнут. Поскольку требования к вычислительной технике постоянно растут, многие страны вкладывают значительные средства в исследовательскую работу по созданию ЭВМ нового поколения. Отметим, что машины пятого поколения, по-видимому, будут существенно отличаться от распространенных в настоящее время ЭВМ.

1.2. ОСНОВНЫЕ ЗАДАЧИ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Ранее уже было указано на то, что нельзя дать исчерпывающее определение ИИ. Однако можно перечислить задачи, методы решения которых на ЭВМ принято связывать с понятием ИИ. Ниже приводятся краткие характеристики основных таких задач.

Автоматическое решение задач представляет собой не столько вычислительную процедуру поиска ответа, как, например, расчет квадратного корня, сколько нахождение метода решения поставленной задачи. Системы, осуществляющие построение вычислительной процедуры, называют автоматическими решателями задач.

Под распознавателями подразумевают устройства, реагирующие на внешнюю среду через различные датчики, например телекамеры, и позволяющие решать задачи распознавания образов. В таких устройствах результаты распознавания выводятся

на экран или печатающее устройство, а в более современных системах есть возможность синтезировать речевые ответы. Использование ИС с большой или сверхбольшой степенью интеграции позволяет строить системы распознавания речевых команд.

Системы распознавания естественной речи позволяют пользователю упростить взаимодействие с ЭВМ с помощью специализированных языков высокого уровня, близких к естественным.

Задачи доказательства теорем и обучения (например, для овладения навыками в какой-либо игре) решаются с помощью автоматического совершенствования алгоритма посредством обработки пробных вариантов, т. е. как бы с помощью накопления «собственного опыта». Следует отметить, что способность к обучению представляет собой одно из основных свойств ИИ.

В настоящее время многие отождествляют понятия ИИ и экспертных систем. Это отождествление появилось во многом благодаря разработкам по созданию программного и аппаратного обеспечения для ЭВМ пятого поколения, разрабатываемых в рамках упомянутого выше японского проекта. Существующие экспертные системы включают в себя огромные базы знаний, сформированные с помощью информации, получаемой от экспертов, т. е. специалистов в той области, для которой создавалась каждая система. Манипуляция накопленными данными осуществляется в другой части экспертных систем, содержащей правила вывода. Сейчас такие системы с успехом используются в медицине, геологии, а также при проектировании ЭВМ.

Обычные языки программирования не очень удобны для разработки систем ИИ. Для построения таких систем больше подходят такие языки, как Пролог, имеющий встроенный механизм логического вывода, или Лисп, ориентированный на обработку списков. Кроме того, создано множество специализированных языков, позволяющих решать ряд отдельных задач ИИ. Пролог был изобретен в Европе и вскоре, к удивлению многих специалистов, был выбран основным языком в японском проекте создания ЭВМ пятого поколения. Язык Лисп, распространенный преимущественно в США, начинает постепенно вытесняться Прологом.

Для эффективной работы мощных систем ИИ необходима высокая скорость доступа к большим базам данных, а также высокое быстродействие. ЭВМ с обычной архитектурой не удовлетворяют этим требованиям. Обычные последовательные методы решения задач постепенно уступают место методам параллельной обработки, когда несколько процессоров независимо друг от друга выполняют различные части одной программы. В настоящее время фирма Intel приступила к выпуску микросхем, названных транспьютерами. Использование этих устройств позволяет решить проблему распараллеливания на аппаратном уровне. В ближайшее время, очевидно, аналогичные устройства начнут выпускать и другие фирмы.

Ряд современных разработок направлен на создание аппаратных средств реализации трансляторов с языков логического программирования, в том числе и с языка Пролог. Скорость работы систем ИИ в последнее время стали выражать с помощью новых единиц измерения — липсов (LIPS — logical inferences per second), обозначающих число логических выводов в секунду. В настоящее время созданы ЭВМ, способные работать с быстродействием в несколько сотен липсов. Однако такое быстродействие нельзя считать удовлетворительным, и оно должно быть поднято до миллиона липсов и выше.

В системах искусственного интеллекта человеческие знания, необходимые для решения задач ИИ, должны быть представлены и записаны в форме, пригодной для последующей обработки на ЭВМ. Сложность заключается в том, что многие аспекты знаний изменяются в зависимости от условий и с трудом поддаются описанию, оставаясь при этом очевидными для человека. Знания должны храниться в системах ИИ в некоторой обобщенной для данной предметной области форме, позволяющей использовать выбранное представление в любой возможной ситуации. Для хранения знаний требуется большая область памяти, и, кроме того, значительное время уходит на их предварительную обработку. Знания, заложенные в систему ИИ, должны быть понятны человеку. Это очевидное условие может быть упущено при разработке системы. С другой стороны, знания должны представляться в форме, удобной для обработки на ЭВМ.

Многие аспекты ИИ связаны с развивающейся в настоящее время наукой — робототехникой. Идея создания «разумного» робота, способного «учиться на собственном опыте», представляет собой одну из центральных проблем ИИ. Такой робот может обладать способностью к ведению диалога на ограниченном естественном языке и уметь решать задачи, требующие инициативы и некоторой оригинальности мышления. Для этого необходимо некоторое предварительное обучение робота, в результате которого он мог бы в отличие от используемых сейчас промышленных роботов выполнять целенаправленные и заранее незапрограммированные действия.

В течение многих лет идеи ИИ серьезно не рассматривались. Это происходило отчасти благодаря чрезмерному оптимизму некоторых теоретиков, а также из-за появления ряда сенсационных публикаций по этому предмету, впоследствии оказавшихся во многом несостоятельными. Идея аппаратно-программных моделей человеческого мозга вызывала насмешки, а в сфере технического производства стали избегать разработок, связанных с ИИ, так как результаты их внедрения явно не соответствовали ожиданиям. Эта в полном смысле слова плачевная ситуация в настоящее время изменилась к лучшему благодаря новейшим достижениям в разработке аппаратуры и программного обеспечения ЭВМ.

Однако сложившееся к настоящему времени взаимное недоверие между теоретиками ИИ и представителями промышленной сферы может привести к тому, что научно-исследовательские работы в данной области будут испытывать значительные финансовые трудности, и, кроме того, эти работы будут оторваны от реальных нужд промышленности. Следует обратить внимание на опыт индустриального развития Японии, добившейся за последние 20 лет рекордных достижений в экономике. Такой успех стал возможен благодаря быстрому и повсеместному внедрению результатов научных исследований.

Эта книга рассчитана на студентов и непрофессионалов, обладателей бытовых персональных ЭВМ (ПЭВМ). Мы будем избегать задач, для решения которых нужна сложная и доступная лишь профессиональным программистам техника, а также берем на себя заботу о том, чтобы читатель увидел реальную возможность применения полученных сведений на практике. Изучение возможностей своего домашнего компьютера — интересное и приятное занятие. При этом многие фундаментальные концепции ИИ усваиваются легче, чем с помощью универсальных ЭВМ, общение с которыми достаточно усложнено. Конечно, не стоит преувеличивать возможности микроЭВМ. Невозможно, например, на бытовых ЭВМ достигнуть такой же скорости логического вывода, как на универсальных. Так же бесполезны попытки создания экспертной системы на компьютере, имеющем объем оперативной памяти 2К. Следует трезво оценивать возможности бытовой вычислительной техники.

В следующем разделе коротко рассмотрим некоторые реальные системы ИИ как сложные, так и достаточно простые.

1.3. НАИБОЛЕЕ ИЗВЕСТНЫЕ СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

ЭЛИЗА (ELIZA). Почти в каждой работе по ИИ можно встретить упоминание о программе ЭЛИЗА, написанной в конце 60-х годов профессором Массачусетского технологического института Дж. Вайзенбаумом. Эта программа была создана для анализа фраз на естественном языке и была названа по имени главной героини пьесы Б. Шоу «Пигмалион», которую, как известно, по ходу действия пьесы ученый-языковед на паре учит правильной английской речи. Основная мысль Шоу заключалась в том, что Элиза до этого курса обучения и после него оставалась тем же самым человеком, т. е. с одними и теми же добродетелями и пороками, хотя окружающие воспринимали девушку-цветочницу, разговаривающую на примитивном жаргоне и прекрасно одетую даму с изысканной речью, как совершенно разных людей. Можно высказать аналогичную мысль и об ЭВМ, остающейся безжизненной совокупностью электронных устройств, несмотря на сложные

программы, загружаемые в нее. Вайзенбаум [3] подробно описывает организацию этой программы и показывает, как ее следует использовать. ЭЛИЗА может «научиться» вести диалог на любую тему, располагая небольшим набором ключевых слов, выбранных случайным способом из исходного предложения, введенного собеседником, а затем с помощью слов, получаемых от него в процессе диалога. Слова из сообщения, введенного пользователем, последовательно сравниваются с ключевыми словами, находящимися в стеке. Затем из стека возможных ответов выбирается фраза, содержащая данное слово. Если во входной строке ключевое слово не было найдено, ответ выбирается случайным образом из другого стека, содержащего нейтральные общие фразы, которые обычно присутствуют почти в каждой беседе и при этом совершенно не влияют на ее развитие. Такие фразы можно обнаружить не только в пустой болтовне, но и в деловом разговоре. Таким образом, ЭЛИЗА может поддерживать разговор на любую предложенную собеседником тему.

Создавая программу ЭЛИЗА, Вайзенбаум не ставил своей целью разработку системы ИИ. На примере данной программы он хотел продемонстрировать сложность общения с ЭВМ на естественном языке, так как диалог между людьми с трудом поддается точному определению и никогда заранее нельзя предугадать, какие смысловые оттенки придают собеседники тем или иным фразам. К удивлению автора программы ЭЛИЗА один из ее вариантов был воспринят серьезно. Он назывался ДОКТОР (DOCTOR) и был задуман Вайзенбаумом как пародия на одну из широко известных школ психотерапии. Во время «психотерапевтического сеанса» ДОКТОР задает вопросы (типа «Каким образом?» или «Почему вы об этом говорите именно сейчас?») или в другом случае повторяет слегка перефразированный ответ собеседника, приглашая его тем самым развивать свою мысль дальше. Вайзенбаум никак не ожидал, что эта шуточная программа будет воспринята как шаг к доказательству возможности общения с ЭВМ на естественном языке и что даже достаточно рассудительные люди всерьез начнут относиться к возможности разрешения своих сугубо личных проблем, посвящая машину в собственные секреты и ожидая от нее помощи как от умудренного опытом врача-психотерапевта.

Возможно, кто-то будет сомневаться в искренности скептического отношения Вайзенбаума к своей программе. Я бы порекомендовал скептикам, а также всем, кого заинтересовали возможности программы ЭЛИЗА, обратиться к книге этого автора и составить собственное мнение о программе. На самом деле различные варианты этой программы получили широкое распространение, и часть психотерапевтов считает, что она может помочь в диагностике и даже в лечении ряда психических нарушений. Один из таких сторонников программы ЭЛИЗА предложил установить ЭВМ с этой программой в общественных местах, таких, как,

например, вокзалы и почтовые отделения, чтобы каждый имел возможность обратиться к машине, заплатив несколько долларов за сеанс.

Читатель может написать собственный вариант программы ЭЛИЗА для бытовой ПЭВМ, но, конечно, при условии, что у нее достаточный объем оперативной памяти. Однако можно привести несколько причин, объясняющих, почему заниматься программированием еще одного варианта ЭЛИЗЫ не следует. Во-первых, алгоритм, реализованный программой ЭЛИЗА, используется в современных системах ИИ достаточно редко. Данная программа была разработана для создания иллюзии «разумности машины», а не для моделирования каких-либо мыслительных способностей человека. Кроме того, чтобы написать и отладить такую программу, понадобится много времени и можно считать, что это время будет потрачено впустую, поскольку, как уже было отмечено, методы программирования, используемые при ее написании, для создания систем ИИ не применяются. Правда, необходимо отметить, что ЭЛИЗА в свое время наделала много шума. Но следует при этом учитывать, что тогда возможности вычислительной техники были известны лишь небольшой группе специалистов. В наше же время, когда школьники с помощью собственных программ проникают в засекреченные базы данных сложнейших систем, а ЭВМ стала предметом домашнего обихода, такая программа едва ли привлечет много внимания.

Система MYCIN. Любая область деятельности всегда опирается на большой объем знаний, полученных опытным путем. Процесс обучения специалиста обычно состоит из двух основных периодов: сначала знания накапливаются, а затем проверяются на практике. Фактический материал составляет значительную часть знаний специалиста, но важную роль играют также различные неформальные правила, позволяющие строить правильные решения на основе неполной информации. Поэтому и для ученика, постигающего ремесло, и для студента, изучающего какую-либо область науки, одинаково важно не только приобрести основные знания по интересующему его предмету, но и узнать о неточных, неопределенных до конца его сторонах.

Факты заносятся в экспертную систему непосредственно в том виде, в каком они обычно представляются в печатных документах. Гораздо сложнее в этих системах формируется часть, реализующая моделирование экспертизы. Эта часть состоит из формализованных правил, которыми руководствуется эксперт в своей работе. Обычно экспертные системы строятся из двух основных частей: базы знаний и программного механизма вывода, выполняющего действия, основанные на логической дедукции. Указанный механизм с помощью заложенных в него правил и на основании имеющихся фактов строит логические заключения, иногда дополняя ответ оценкой его правдоподобия.

Автор программы MYCIN Е. Шортклиф описал ее в 1976 г. Эта программа предназначалась для консультационной помощи при диагностике инфекционных заболеваний крови. Врачи обращались к указанной программе за консультацией при определении болезни пациента и могли сами обновлять первоначально заложенную информацию, вводя новые данные и правила.

Перечислим основные свойства программы MYCIN. Во-первых, эта программа может с помощью правил вывода и на основании данных, заложенных изначально и вводимых пользователем, делать логические выводы. Во-вторых, в программу MYCIN заложены средства, позволяющие оценивать правдоподобность полученного вывода, пользуясь шкалой оценок от 0 до 1,0. Третье важное свойство заключается в том, что данная программа может давать пояснения к полученному решению, указывая шаг за шагом ход своих «рассуждений». Наконец, в-четвертых, она может быть использована для диагностики других инфекционных заболеваний. Последнее свойство является на наш взгляд наиболее важным. Оно означает возможность замены правил и данных, связанных с диагностикой инфекционных заболеваний, на другие, соответствующие иным областям применения.

Таким образом, можно рассматривать MYCIN как универсальную экспертную систему. Механизм вывода программы MYCIN, называемый EMYCIN (Essential MYCIN или Engine MYCIN), используется в совершенно различных областях деятельности. Интересно, например, применение EMYCIN совместно с программой MARK, предназначенной для решения задач методом конечных элементов. Правильное обращение с этой программой, разработанной по заказу ВВС США, требует большой предварительной подготовки. На основе EMYCIN была создана программа SACON (Structural Analisis Consultant), формирующая обращения к программе MARK с помощью анализа запросов пользователя, написанных на несложном языке структурных спецификаций.

Еще один пример использования программы EMYCIN — пакет ГУИДОН (GUIDON), разработанный в Станфордском университете В. Кланси и описанный в его докторской диссертации в 1979 г. С помощью этого пакета можно создавать программы для обучения работе с любой основанной на EMYCIN экспертной системой. Указанный пакет программ использует специальные правила для создания методики обучения и правила вывода для выбранной предметной области. С помощью программы EMYCIN можно создавать обучающие программы для самых разных областей приложения. В последней главе мы рассмотрим несколько небольших экспертных систем, основанных на ряде принципов, используемых в программах EMYCIN и ГУИДОН.

ДЕНДРАЛ (DENDRAL). В функционировании популярной экспертной системы ДЕНДРАЛ, используемой для решения задач химического анализа, можно выделить три основных этапа. Сначала с помощью базы знаний составляется список исходных

условий, дополняемый на втором этапе пользователем, затем система генерирует, проверяет и ранжирует возможные решения, после чего выводит их на печать в порядке рангов. Первоначально ДЕНДРАЛ была написана на языке Лисп, но затем была реализована на других языках, что позволило перенести ее на различные ЭВМ. Одна из таких версий была создана в г. Эдинбурге для PDR-10. Основные этапы работы данной версии — это планирование, генерация и проверка решений. Указанная версия системы ДЕНДРАЛ используется в США и в Западной Европе и представляет собой едва ли не лучшее на сегодняшний день применение системы ИИ.

В статье профессора Станфордского университета Е. Фейгенбаума, напечатанной в сборнике «Интеллектуальные системы» под редакцией Д. Хейеса и Д. Мичи [6], указаны два главных принципа инженерии знаний. Согласно первому для построения и функционирования любой экспертной системы главными являются закладываемые в систему знания. Недостаточный объем знаний не может быть компенсирован более сложным механизмом вывода, хотя долгое время многие придерживались противоположной точки зрения.

Второй не менее важный принцип заключается в том, что большая часть знаний должна носить эвристический характер, т. е. представлять часть человеческого опыта, позволяющую специалистам в какой-либо области находить правильные решения в условиях с неполной информацией. Так, например, опытный автомеханик может сразу обнаружить большую часть неисправностей мотора, однако при этом не всегда в состоянии объяснить, какие именно факторы он использовал при диагностике. Эксперт пользуется всеми своими природными чувствами совершенно бессознательно, не отдавая себе отчета, каким образом каждое из них влияет на получение ответа.

О важности знаний для экспертных систем в первую очередь должны помнить пользователи персональных ЭВМ, так как боязнь создать программу, не уместящуюся в памяти машины, может привести к попытке улучшить систему с помощью многократного совершенствования механизма вывода в ущерб пополнению базы знаний.

1.4. ИНЖЕНЕРИЯ ЗНАНИЙ

Для построения современных систем ИИ необходимы новые методы построения баз знаний. Как было указано в предыдущем разделе, многие аспекты знаний нельзя точно определить, поэтому при создании баз знаний основная задача заключается в получении необходимых знаний от экспертов и вводе их в ЭВМ так, чтобы потом они могли быть использованы системой. Главная трудность здесь состоит в том, что формирователь базы знаний

(инженер знаний), как правило, не является специалистом в той области деятельности, для которой создается экспертная система. В то же время эксперт не всегда может быть достаточно знаком с методами программирования баз знаний. Кроме того, свои знания эксперт использует чаще всего бессознательно и не всегда может, даже если и хочет, передать все свои знания и навыки.

Как же должен поступать в таких случаях инженер знаний, какие задавать вопросы эксперту? Поиском ответов на такие вопросы занята новая наука — инженерия знаний. Основные ее усилия направлены на создание систем, способных автоматизировать большую часть работ по формированию базы знаний. Хорошим примером таких систем служит упомянутая выше система SACON. Поскольку потребность в инженерах знаний возникла недавно, таких специалистов в настоящее время мало, к тому же почти все они заняты исследовательской работой в различных университетских центрах. Однако очевидно, что область применения экспертных систем быстро расширяется и в ближайшее время количество инженеров знаний, работающих в сфере производства, значительно возрастет. С другой стороны, необходимость в таких специалистах пропадет, когда будет решена задача автоматизации формирования знаний.

В упоминавшейся выше книге «Интеллектуальные системы» [6] есть статья под названием «Прототип метода очистки знаний», написанная Мичи. В ней описывается методика получения знаний из исходного материала, чем-то напоминающая способ выделения чистого бензина из нефти. Исходный материал для базы знаний может быть получен от экспертов или взят из различных печатных источников, например из книг, технической документации и т. д. В статье Мичи приводится ряд аргументов в пользу того, что Англия в ближайшее время будет играть значительную роль в производстве систем ИИ, так как именно в этой стране несколькими учеными, работающими в области информатики, были получены результаты, которые с успехом могут быть использованы при создании систем ИИ, пригодных для промышленного производства. Большинство этих ученых, не имея возможности заняться промышленной реализацией своих разработок, обратилось за помощью к соответствующим фирмам.

По сравнению с другими видами производства финансовые затраты на реализацию систем ИИ довольно низки, так как многие фирмы, специализирующиеся на изготовлении программного обеспечения, имеют в своем штате достаточное число квалифицированных специалистов, способных заняться разработкой указанных систем. Вполне возможно, что Англия могла бы стать поставщиком специалистов по системам ИИ.

Однако далеко не все проблемы решены и обсуждение возможностей промышленного производства таких систем сейчас в самом разгаре.

1.5. ДРУГИЕ ЭКСПЕРТНЫЕ СИСТЕМЫ

Проф. Мичи связан с фирмой Intelligent Terminals, разработавшей экспертную систему Expert-Ease («Эксперт-Из»), которая распространяется фирмой Expert Software International. Большинство коммерческих экспертных систем имеют узкую специализацию, к тому же, довольно дороги, поэтому их реализацией могут заниматься только достаточно крупные фирмы, специализирующиеся на продаже таких систем. По сравнению с другими пакетами программ для микроЭВМ система Expert-Ease довольно дорога, однако ее цена невысока. К тому же, пользователь может приобрести готовую систему за более низкую плату. Expert-Ease — универсальная система (такие системы называют также оболочками экспертных систем). С ее помощью можно строить экспертные системы для различных областей приложения, а также пакеты графических программ, текстовые редакторы и игровые программы для домашних персональных ЭВМ. Первая версия Expert-Ease была разработана для персонального компьютера IBM PC, обладающего оперативной памятью емкостью 128К и снабженного двумя двусторонними накопителями на гибких дисках, каждый емкостью 320К. Вместе с оболочкой системы поставляется также демонстрационный пример, позволяющий пользователю быстрее научиться работать с оболочкой.

Работа системы Expert-Ease заключается в индуктивном выводе нужного правила из достаточно большого числа образцов, а затем в получении результата с помощью этого правила. Тип получаемого ответа зависит от области применения системы. Прилагаемый к поставляемой системе демонстрационный пример представляет собой экспертную систему для обнаружения неисправностей электронной схемы с помощью 20 тестов. Когда процедура проверки осуществляется с помощью обычного руководства по тестированию данной схемы, необходимо использовать все 20 тестов, в результате чего констатируется исправность схемы, или удается обнаружить какую-либо комбинацию ошибок. Аналогичная проверка с помощью экспертной системы требует всего шести тестов для того, чтобы убедиться в исправности схемы, и самое большее трех тестов для обнаружения любой из неисправностей. Каждый специалист по радиоэлектронике сможет оценить экономичность такого метода тестирования.

Несмотря на то, что система обладает весьма разнообразными возможностями, научиться обращаться с ней довольно просто и для работы с ней знаний по программированию не требуется. Пользователь вводит в систему названия атрибутов, которые описывают рассматриваемый объект. Общее число атрибутов не должно превышать 31. Каждый атрибут может быть символьным или числовым. Каждый числовой атрибут может принимать одно из 255 значений, заранее подобранных из числового диапазона от $-32\,766$ до $+32\,766$. Длина символьных атрибутов ограничи-

Таблица 1.1

Лист	Стебель	Корень	Семена	Цвет- ки	Окраска	Тип растения
Широкий	Высокий	Стержневой	Споры	Есть	Желтая	Сорняк
Широкий	Стелю- щийся	Стержневой	Тяжелые	Нет	Нет	Сорняк
Строенный	Короткий	Мочковатый	Легкие	Есть	Белая	Сорняк
Узкий	Короткий	Мочковатый	Легкие	Нет	Нет	Злак

вается десятью. Ввод значений атрибутов осуществляется с помощью таблицы, в которой каждый столбец соответствует какому-либо атрибуту. Последний столбец таблицы обычно отводится для результата диагностики. В период обучения системы сначала пользователь сам заполняет последний столбец, после чего система обрабатывает введенные наборы значений и пытается найти связывающее их правило с помощью построения дерева вывода. После обучения системы данное дерево используется для получения результатов по вводимым значениям атрибутов. В ряде случаев система не может найти нужное правило вывода. Это может произойти, например, из-за выбора неудачных примеров при обучении или в том случае, если какие-либо примеры противоречат один другому. В таких ситуациях система сообщает пользователю причину неудачи и ожидает в первом случае ввода дополнительных примеров, а во втором — исправления примеров, находящихся в конфликте.

Символьные атрибуты могут задавать бинарные логические величины, например «да» и «нет», или в другом случае определять значения нечеткой логики, такие, как, например, «длинный», «короткий», «зеленый», «серый» и т. д. Пользователь, если это ему удобнее, может выражать значения атрибутов в числовом виде. В табл. 1.1 представлен пример определения значений атрибутов.

При решении практических задач требуется умение различать до нескольких сотен различных видов сорняков и злаков с помощью гораздо более длинного списка атрибутов. Например, для более подробного описания семян могут быть заданы дополнительные атрибуты: окраска, размер и т. д. Ответы, как было указано, помещаются в правый столбец «Тип растения».

Для всех атрибутов составляются вопросы, с помощью которых заполняются соответствующие графы таблицы. Диалог системы с пользователем осуществляется следующим образом. Когда требуется задать значение очередного атрибута, его название отмечается пользователем с помощью курсора в соответствующей таблице названий на экране дисплея. После этого на экране появляется вопрос, позволяющий правильно задать значение данного атрибута. Так, например, атрибуту «окраска» из при-

ведения выше примера может соответствовать вопрос: «Какая окраска цветков этого растения?» Если пользователь не может ответить на какой-либо из вопросов, он вводит символ неопределенности — «X». Таким образом управление диалогом осуществляется в основном пользователем, что обеспечивает большие удобства при эксплуатации системы.

Теперь рассмотрим две другие системы — АМ и Eurisco («Эвриско»), разработанные в Станфордском университете д-ром Д. Ленатом для исследовательских и учебных целей (в отличие от Expert-Ease, созданной для коммерческой эксплуатации). Однако с недавнего времени разработка этих систем финансируется 22 американскими фирмами, поддерживающими все перспективные проекты, которые могут служить ответом на японский проект создания ЭВМ пятого поколения.

Автор указанных систем Ленат считает, что эффективность любой экспертной системы определяется закладываемыми в нее знаниями. По его мнению, чтобы система была способна к обучению, в нее должно быть введено около полумиллиона сведений общего характера. Это примерно соответствует такому объему информации, каким располагает четырехлетний ребенок со средними способностями. Ленат также считает, что путь создания узкоспециализированных экспертных систем с уменьшенным объемом знаний ведет к тупику.

В систему АМ первоначально было заложено около 100 правил вывода и более 200 эвристических алгоритмов обучения, позволяющих строить произвольные математические теории и представления. Сначала результаты работы системы были весьма многообещающими. Она могла сформулировать понятия натурального ряда и простых чисел. Кроме того, она синтезировала вариант гипотезы Гольдбаха о том, что каждое четное число, большее двух, можно представить в виде суммы двух простых чисел. До сих пор не удалось ни найти доказательства данной гипотезы, ни опровергнуть ее. Дальнейшее развитие системы замедлилось и было отмечено, что, несмотря на проявленные на первых порах «математические способности», система не может синтезировать новых эвристических правил, т. е. ее возможности определяются только теми эвристиками, что были в нее изначально заложены.

При разработке системы Eurisco была предпринята попытка преодолеть указанные недостатки системы АМ. Как и в начале эксплуатации АМ, первые результаты, полученные с помощью Eurisco, были эффективными. Сообщалось, например, что система Eurisco может успешно участвовать в очень сложных играх. С ее помощью в военно-стратегической игре, проводимой ВМФ США, была разработана стратегия, содержащая ряд оригинальных тактических ходов. Согласно одному из них, например, предлагалось взрывать свои корабли, получившие повреждения. При этом корабли, оставшиеся неповрежденными, получают необходимое пространство для выполнения маневра.

Однако через некоторое время обнаружилось, что система не всегда корректно переопределяет первоначально заложенные в нее правила. Так, например, она стала нарушать строгое предписание обращаться к программистам с вопросами только в определенное время суток. Таким образом, система Eurisco, так же как и ее предшественница, остановилась в своем развитии, достигнув предела, определенного в конечном счете ее разработчиком.

В настоящее время доктор Ленат во главе исследовательской группы занят кодированием и вводом нескольких сот тысяч элементов знаний, необходимых, по его мнению, для создания действительно «интеллектуальной» системы. Этот проект назван Сус («Цик») (от английского слова encyclopaedia).

1.6. ЯПОНСКИЙ ПРОЕКТ СОЗДАНИЯ ЭВМ ПЯТОГО ПОКОЛЕНИЯ

Работы по проекту создания ЭВМ пятого поколения в Японии координируются Технологическим институтом ЭВМ нового поколения (ICOT). В 1987 г. окончился пятый год этого проекта, рассчитанного в общей сложности на десять лет. Вместе с проектированием аппаратуры разрабатывается и программное обеспечение для этих ЭВМ. Соответствующая группа занята созданием языковой среды для новых машин, включающей в себя языки операционных систем, естественно-языковый интерфейс с пользователем ЭВМ, а также языки представления данных. Другая группа занята вопросами применения новых ЭВМ; она отвечает за разработку систем управления реляционными базами данных, экспертных систем, а кроме того, оценивает всевозможные предложения по использованию новых ЭВМ. В целом японские разработчики пытаются сформулировать единую унифицированную теорию представления и обработки информации.

В настоящее время при решении задач, относящихся к различным областям ИИ, применяются разные подходы. Однако, если бы удалось, как надеются японские специалисты, найти общую теорию, объединяющую все эти подходы, поиск нужного метода решения можно было бы значительно упростить, сократив число вариантов при выборе. В результате четырехлетнего периода работ в рамках проекта была создана ЭВМ Delta («Дельта»), функционирующая с помощью обработки элементов базы знаний. В этой машине присутствуют такие компоненты, как система логического вывода, база знаний, естественно-языковый интерфейс, а также система создания программного обеспечения с помощью прототипов.

Реляционная база данных, используемая в ЭВМ Delta, имеет объем около 20 Гбайт. Для работы с этой базой данных на начальных этапах создания компьютера Delta использовались методы

последовательного поиска, принятые в обычных ЭВМ. Вынужденный компромисс показал необходимость максимального распараллеливания операций с базами данных. По оценкам специалистов производительность программно-аппаратных средств вывода должна в окончательных вариантах ЭВМ Delta достигнуть миллиарда липсов. Для такого быстродействия понадобится 100 одинаковых одновременно работающих процессоров. Полагают, что ряд японских фирм сможет уже в начале 90-х годов освоить производство таких интегральных схем.

1.7. ЗАМЕЧАНИЯ ДЛЯ НАЧИНАЮЩИХ

Владельцы персональных ЭВМ, прочитав предыдущие разделы и ознакомившись с размерами затрат, необходимых для построения систем ИИ, могут решить, что, поскольку аппаратура для реализации систем ИИ слишком дорога и доступна лишь крупным фирмам, нет смысла пытаться создавать такие системы самостоятельно. Можно привести несколько аргументов против такой точки зрения. Во-первых, как показала практика, многие из разработок, выполненных вчера на самом высоком уровне, сегодня становятся доступными массовому пользователю. Результаты исследований влияют на нашу работу, нашу жизнь и даже на наши мысли. Во-вторых, быстродействие вычислительных машин все время растет, и те программы, которые ранее могли быть использованы лишь на очень мощных ЭВМ, теперь применяются на обычных машинах.

Третий и, наверное, самый главный аргумент заключается в том, что мы всегда можем при некоторых ограничениях попробовать на практике любые, даже самые передовые методы на небольших персональных ЭВМ. Например, известно, что японцы используют в своем проекте Пролог и различные варианты как основные языки. Однако этот факт не может служить препятствием для желающих изучить язык Пролог, понять его философию и испытать его в действии на своих ЭВМ, даже если эти машины по своим техническим характеристикам могут поддерживать лишь небольшую учебную систему ИИ.

Одним из первых практических результатов, полученных японскими специалистами в ходе работ по проекту создания ЭВМ пятого поколения, была разработка машины PSI («Пси») — персональной ЭВМ последовательного вывода, предназначенной для исследования логических методов решения задач. Программное обеспечение для данной ЭВМ, возможно, имеет более важное значение, чем аппаратура: в языке программирования предусмотрена реализация методов логического программирования, а операционная система Simpos («Симпос») использует расширенную версию языка Пролог — ESP (Extended Self-contained Prolog). ESP позволяет создавать удобные многооконные объектно-ориен-

тированные среды для решения задач логического программирования. Быстродействие опытных образцов PSI, равное приблизительно 30 лнпсам, по мнению специалистов, может быть повышено при подготовке этой ЭВМ к промышленному производству до 200 лнпсов. Возможно, что эта машина будет первой доступной широкому кругу пользователей ЭВМ нового поколения.

В заключение — несколько слов относительно представленных в данной книге примеров, иллюстрирующих методы логического программирования на языке Пролог. Такие примеры достаточно просты и коротки: они редко превышают страницу печатного текста. Но эти фрагменты программ позволяют читателю, с одной стороны, лучше понять главные идеи языка Пролог, а с другой, в случае необходимости, дополнить их в соответствии с возможностями имеющихся в их распоряжении ЭВМ. Кроме того, анализ примеров позволяет понять основные принципы построения реляционных баз данных и экспертных систем. При изложении материала автор не старался сделать его академически строгим. Его цель — вызвать интерес читателя к методам ИИ, воодушевить его на собственные эксперименты в этой области. В случае необходимости читатель может расширить полученные знания с помощью специальной литературы по ИИ.

2.1. ПОЧЕМУ СЛЕДУЕТ ИЗУЧАТЬ ПРОЛОГ?

Существует довольно много причин, позволяющих объяснить, почему пользователи ЭВМ (как новички, так и профессионалы) должны уметь работать с языком Пролог. В настоящее время Пролог — это широко известный язык с хорошей документацией, который используется в Японии как базовый язык для ЭВМ пятого поколения. Крупные денежные средства были вложены в разработку Пролога в рамках программы «Эсприт» Европейского экономического сообщества и проекта «Элви» в Великобритании, а также министерством обороны США и компанией IBM.

Пролог обладает мощными средствами, позволяющими извлекать информацию из баз данных, причем методы поиска данных, используемые в нем, принципиально отличаются от традиционных. Мощь и гибкость баз данных Пролога, легкость их расширения и модификации делают этот язык очень удобным для коммерческих приложений. Все сказанное выше, а также достаточная простота языка для изучения вселяют уверенность в том, что не только профессиональные программисты, но и люди, не работавшие ранее на ЭВМ, пристрастятся к Прологу. Название Пролог (PROLOG) образовано из первых частей английских слов PROgramming и LOGic. Пролог был создан проф. А. Колмеро, который в начале 60-х годов был привлечен к исследованиям в рамках проекта по методам быстрого обнаружения синтаксических ошибок в программах. Эти исследования в 1963 г. привели к созданию программных средств анализа естественных языков, и, в конце концов, в Марсельском университете была разработана первая версия языка Пролог. Сначала Пролог получил известность только во французских академических кругах и был ориентирован на архитектуру французских ЭВМ. Но вскоре Прологом заинтересовалась фирма DEC. Для ЭВМ этой фирмы сравнительно быстро был реализован транслятор с Пролога и в результате Пролог прочно занял место в ряду коммерческих программных средств.

В Великобритании аналогичные работы, вызвавшие большой интерес во всем мире, проводились проф. Р. Ковальским и коллективом исследователей лондонского имперского колледжа. Следует отметить и другие разработки: реализацию Пролога на DEC-10 в Эдинбургском университете, новую работу ученых

Марсельского университета, выполненную в 1982 г., венгерскую версию языка, названную М-Пролог, и, конечно, Японский проект.

Версии Пролога, доступные непрофессионалам, работают под управлением операционных систем CP/M, MSDOS и UNIX. В этой книге использована версия языка, разработанная Ассоциацией логического программирования для микроЭВМ ZX Spectrum фирмы Sinclair. Последователи, работающие с другими системами, будут испытывать незначительные трудности, поскольку остальные реализации Пролога отличаются от используемой лишь в деталях. Хотя объем доступной оперативной памяти и ограничивает размеры разрабатываемых программ, в данном случае это не мешает конструировать довольно сложные программы. Для начинающих практика использования языка Пролог на Spectrum или аналогичной ЭВМ послужит введением в современные языки, ориентированные на решение задач искусственного интеллекта. Очевидно, что размеры программ менее важны, чем те понятия, которые могут быть с их помощью изучены. В любом случае большинство людей по мере накопления опыта работы на ЭВМ начнут использовать стандартный синтаксис микроПролога и хочется надеяться, что все они будут покорены достижениями логического программирования, обеспечивающего такое мощное программное обеспечение для персональных ЭВМ.

Здесь не уделяется внимания изучению языка Пролог как такового. Отметим, что этой цели служат многие прекрасные книги, читатели могут узнать их названия из приведенного списка литературы [1—12]. В любом случае автор надеется, что читатели, в распоряжении которых нет компилятора Пролога, все же смогут усвоить основные принципы. Если Вы приобретете систему программирования на языке Пролог для микроЭВМ Spectrum, руководства по микроПрологу для начинающих (оно кстати, входит в комплект вместе с программным обеспечением) Вам будет вполне достаточно для того, чтобы начать работу. Можно также рекомендовать описание микроПролога для ЭВМ Spectrum, подготовленное в Кембридже. В заключение остановимся еще на одном моменте. Существует несколько специальных операций, доступных только для пользователей ЭВМ Spectrum, с помощью которых некоторые сложные вычисления можно выполнять довольно просто. О них речь пойдет ниже.

2.2. ОСНОВНЫЕ ПОНЯТИЯ

Сначала нужно установить различие между дескриптивными и императивными языками. Все, кто видел листинги программ в машинных кодах, знают, что, используя только листинг, почти невозможно определить точно, что же программа делает. Правда, определенные соображения все-таки можно высказать. Программа

в машинных кодах представляет собой список инструкций или команд, предписывающих ЭВМ, что делать; понять же, для чего эта программа предназначена, практически невозможно. Для того чтобы получить информацию о цели разработки программы и путях достижения этой цели, необходимо обратиться к документации, которая обычно не входит непосредственно в программу.

Машинный язык представляет собой императивный язык самого низкого уровня. Языки более высокого уровня, такие, как Бейсик, по своей природе носят более описательный характер. По текстам программ, написанных на таких языках, легче определить, для чего программы предназначены, но все же и в этом случае больше информации дают программные комментарии и документация. Дескриптивные языки должны быть настолько самодокументирующимися, насколько это возможно. В этом отношении Пролог — в основном дескриптивный язык, но с императивными элементами.

Если Вас попросят рассказать об ЭВМ кому-то, кто ничего о них не знает, то наиболее разумно начать с того, что машина может делать, а не с того, как она это делает. После того, как первое будет усвоено, можно переходить ко второму. Точно так же обстоит дело и с языками программирования. Императивные языки, подобные Бейсику, состоят из команд, предписывающих ЭВМ, как решить задачу, например, выполнить оператор

FOR X = 1 TO 50

и т. п. Дескриптивные языки позволяют максимально приблизиться к естественному языку, например, в них разрешены конструкции типа «Определить, кто самый богатый человек в городе?»

Прологу наиболее адекватна последняя форма. Правда, в его состав входит несколько императивных конструкций, связанных в основном с выполнением чисто машинных операций. Например, команды LIST, CLEAR и OUT предназначены для распечатки листинга, очистки экрана и выдачи информации. Заметим, что такая команда, как RUN, в Прологе отсутствует. Вместо нее пользователям разрешено создавать базу знаний, определять отношения между ее элементами и формировать запросы либо на извлечение данных из базы, либо на генерацию новых данных. При обработке запросов в Прологе используется специальный встроенный механизм манипулирования данными, опирающийся на те отношения, которые определит пользователь. Поскольку понятие отношения является центральным в Прологе, рассмотрим его с привлечением ряда примеров.

2.3. ОТНОШЕНИЯ

Интуитивно, отношение понимается как правило, связывающее два объекта. Наиболее часто в нашем обиходе фигурируют отношения между людьми, например, высказывание «Том является братом Джеймса» определяет отношение между Томом и Джейм-

сом. Слова «является-братом» связаны дефисом, поскольку в Прологе в качестве имени отношения может использоваться только одно слово. Отношения между двумя объектами, аналогичные приведенному выше, называются бинариями. Ниже даны примеры еще нескольких бинарных отношений.

1. Алекс владеет книгами.
2. Джо любит Кейт.
3. 25 больше 24.

Не все реально существующие отношения являются бинарными. Например, отношения типа «возраст Джеймса» использует только один объект. Такие отношения называются унарными. Существуют отношения и большей ариности. Например, в отношении «Алан дает деньги Брайану» фигурируют три объекта. Но на первом этапе такие сложные отношения нами не будут использоваться. Объекты отношения в литературе известны как аргументы.

В связи с этим мы можем сказать, что бинарное отношение характеризуется двумя аргументами, а унарное — одним. В более общей форме бинарное отношение можно представить в виде XYR , где R — имя отношения, а X и Y — его аргументы. Унарное отношение тогда имеет вид XR , где X — аргумент отношения R .

Заметим, что отношения не являются единственным способом представления знаний. Для этой цели также можно использовать любые типы кодов и хранить их, например, в виде числовых массивов или массивов строк в ЭВМ. Но поскольку данные предназначены для людей, необходимо уметь преобразовывать их в форму предложений достаточно простого языка. Средство, с помощью которого такое преобразование выполняется, называется функцией отображения. Важно, что способ, использующий отношения, позволяет очень просто перейти к обычному человеческому языку. Кроме того, если названия отношений выбраны достаточно продуманно, можно написать программу так, что она будет напоминать текст на естественном языке и необходимость в преобразовании отпадает. В этом случае программисты имеют возможность самостоятельно выбирать те слова, которые будут впоследствии входить в тексты их программ, и не связаны жестко ограниченным набором слов и символов, допускаемых ЭВМ.

Следующая короткая программа, служащая для иллюстрации только что изложенных положений, без сомнения будет понятнее всем, вне зависимости от того, есть у них в распоряжении транслятор с Пролога или нет:

Rover isa dog	; Ровер это собака
Tom isa man	; Том это человек
X has-a-tail if X isa dog	; X имеет-хвост, если X это собака

Эта программа состоит из двух утверждений, представляющих собой факты, и одного правила, с помощью которого она может генерировать другие факты. Программа написана на обычном разговорном языке. Исключением является лишь то, что слова, определяющие отношения, соединены дефисами. Ниже показано, как может протекать процесс работы такой программы:

is (Rover isa dog)	Запрос вводится пользователем
{верно (Ровер это собака)}	
YES	Ответ системы
[ДА]	
all (x: x isa man) --	Еще один запрос
{определить все (X: X это человек)}	
Tom	Ответ системы
[Том]	
No (more) answers	Это сообщение свидетельствует о том,
{Ответов (больше) нет}	что все ответы найдены
is (Rover has-a-tail)	Запрос
{верно (Ровер имеет-хвост)}	
YES	Ответ
[ДА]	
is (Tom has-a-tail)	Запрос
{верно (Том имеет хвост)}	
NO	Ответ
[НЕТ]	

Эта программа служит для демонстрации возможностей Пролога и практической ценности не имеет. Анализируя работу программы, можно прийти к следующим выводам. Во-первых, программа, обрабатывая факты, может генерировать сообщения YES/NO (или TRUE/FALSE) в зависимости от того, являются ли заданные объекты аргументами отношения. Во-вторых, она может осуществлять поиск значений всех тех аргументов, которые вместе с заданными образуют указанные отношения. Для этого используются так называемые *несвязанные* переменные, которые не имеют постоянных значений в программе. Конкретные значения связываются с этими переменными только при поиске ответа на запрос.

Два последних запроса дают возможность понять, как в Прологе производится логический вывод, позволяющий генерировать новые факты. Логический вывод в Прологе основан на использовании как фактов из базы данных, так и логических правил. В Прологе реализован классический механизм умозаключений. Например, из двух посылок

«Все собаки имеют хвосты»;

«Ровер — это собака»

следует заключение

«Ровер имеет хвост».

Этот пример может кому-то показаться довольно простым, но именно он позволяет понять, какие возможности открывает

использование Пролога в таких областях, как создание и эксплуатация реляционных баз данных, доказательство теорем, принятие решений и анализ естественных языков.

Второй тип умозаключений, осуществляемых программами, написанными на Прологе, является наиболее спорным. Например, в программе может использоваться следующая цепочка рассуждений.

Из посылок

«Все собаки имеют хвосты»;

«Том — это человек».

делается вывод, что

«Том не имеет хвоста».

Последнее высказывание естественно не является истинным. Сделанное заключение логически не следует из посылок. Даже если доказать, что высказывание

«Том — это человек»

позволяет заключить, что

«Том не является собакой»,

то из этого не следует

«Том не имеет хвоста».

Правда, в соответствии с внутренней логикой программы такого рода умозаключения могут быть разрешены. Например, будем считать, что в программу включены все возможные знания об объектах. Тогда, поскольку нет никакой информации о том, что Том — это собака и известно, что только собаки имеют хвосты, можно сделать вывод

«Том не имеет хвоста»

На практике это не должно привести к сложностям, но знать об этом необходимо, поскольку правило «неудача при доказательстве утверждения ведет к тому, что истинным считается отрицание этого утверждения» используется для реализации логического оператора «not»^{*}.

К более детальному разговору о логических принципах мы еще вернемся, но все же еще несколько слов об этом стоит сказать сейчас. Для нас очевидно, что если Бен — человек, то он не собака, но в программе нет утверждений, на основе которых это заключение может быть получено. Утверждение «Том — не собака» было бы справедливым в том случае, если бы в программе имелось правило

«Нет человека, который был бы собакой».

^{*} Кратко этот принцип называется «Отрицание по неудаче». — Прим. пер.

Аналогично утверждение

«X не имеет хвоста»

не следует из двух следующих утверждений:

«Все собаки имеют хвосты»;

«X не является собакой».

Это можно легко понять, взяв в качестве значения X лошадь или кошку.

В заключение отметим, что такого рода ошибочные логические выводы надо уметь предвидеть и в случае необходимости не допускать.

2.4. СИНТАКСИС ПРЕДЛОЖЕНИЙ ПРОЛОГА

Синтаксис, как известно, — это часть грамматики, в которой анализируется правильность расположения слов в предложении. Стандартный синтаксис Пролога представляет определенные трудности для начинающих и не может быть изучен так же, как синтаксис обычного языка. По этой причине целесообразно использовать более близкую человеку систему, которая переводит предложения обычного естественного языка в предложения стандартного Пролога. На ЭВМ типа Spectrum такая система называется SIMPLE. Ниже приведены некоторые типы предложений, которые допускаются системой SIMPLE:

Простое предложение	Tom isa man [Том это человек]
Условное предложение	Jim owns Spot if Spot white [Джим владеет Спотом, если Спот белый]
Конъюнкция (and)	X isa black-dog if isa dog and X black [X это черная-собака, если X это собака и X черная]
Дизъюнкция (or)	Y isa animal if (either Y isa dog or Y isa cat) [Y это животное если (Y это собака или Y это кошка)]
Отрицание (not)	Joe owns X if X isa dog and not X white [Джо владеет X, если X это собака и X не белая]
isall	X isall (Y: Joe owns Y) [X существует (Y: Джо владеет Y)]
forall	(forall) Joe owns X then X black [(для всех: если Джо владеет X, то X черная)]

Существуют некоторые ограничения, накладываемые на использование трех последних конструкций. В основном это касается условий isall и forall. В связи с этим основное внимание сначала будет уделено первым пяти конструкциям. Система SIMPLE после выдачи команды listall преобразует пять первых предложений к виду

```

Tom isa man
X isa black-dog if
  X isa dog and
  X black
X isa animal if
  (either X isa dog or X isa cat)
Jim owns Spot if
  Spot white
Joe owns X if
  X isa dog and
  not X white

```

Отметим некоторые характерные черты приведенного выше текста.

1. Строки текста не пронумерованы.

2. Строки расположены не в том порядке, в котором они вводились в ЭВМ. В Прологе принято располагать строки, описывающие одно и то же отношение, рядом. Первым было введено предложение, характеризующее отношение «isa»; поэтому далее следуют все предложения, в которых первым употребляется именно это отношение, а затем идут предложения, связанные с отношением «owns». Кроме того, в системе формируются относительные номера предложений, описывающих одно и то же отношение. Например, предложение «Tom isa man» имеет номер 1, поскольку оно является первым, описывающим отношение «isa».

3. В четвертом предложении было использовано имя Y, а после преобразования оно заменено на X. Это может смутить начинающих программистов, особенно если до этого они работали на Бейсике. В Прологе буквы x, y и z резервируются для использования в качестве имен переменных. Напомним, что такие переменные, называемые несвязанными, не принимают в программе конкретных значений до начала обработки запроса. Пролог всегда назначает переменным, входящим в предложение, имена в следующей последовательности: X, Y, Z, x, y, z, X1, X2, X3, x1, x2 и т. д. Для того чтобы у программистов не возникало лишних вопросов, рекомендуем им использовать имена переменных строго в указанном порядке.

4. Весьма важно употребление круглых скобок в предложениях типа

(either isa dog or X isa cat).

Использование скобок делает процесс ввода предложений с более чем одним *ог* (или) неудобным. Например, возникают конструкции вида

(either X isa dog or (either X isa cat or X isa rat)).

Операция *ог* может быть реализована разбивкой одного утверждения с несколькими *ог* на элементарные утверждения, каждое из которых описывает только одно условие. Утверждение с большим числом условий действительно следует разбивать на более мелкие.

5. Система программирования Пролог поддерживает словарь, содержащий информацию обо всех используемых отношениях. Чтобы получить такую информацию на ЭВМ типа Spectrum, достаточно напечатать либо `list dict`, либо `all (x : x dict)`. Можно также определить число утверждений, связанных с каждым отношением; для этого следует использовать `(all x : x dict and x defined)`. Если необходимо просмотреть все утверждения, связанные с каким-то отношением, можно использовать команду `list R`, где `R` — имя отношения. Всю программу можно уничтожить с помощью команды `kill all`, а одно отношение — с помощью команды `kill R`. Для уничтожения одного утверждения следует использовать команду `delete RN`, где `N` — номер утверждения в отношении. Для модификации отношения необходимо ввести команду `edit RN` и обычным путем, подводя курсор в нужное место, внести исправления.

6. Расположение текста программы (листинга) при его выводе на экран организовано следующим образом. После `if` и `and` печать продолжается на следующей строке с отступом в пять позиций. Это приводит к тому, что различные предикаты, относящиеся к одному утверждению, размещаются на разных строках, что, в свою очередь, позволяет подчеркнуть логическую структуру каждого утверждения. В сравнительно редко встречающейся ситуации, когда длина строки превышает допустимую, возможен перенос на следующую строку. Перенесенной части строки всегда предшествуют два пробела. Отметим также, что разделять слова на части (одна на одной строке, другая — на другой) запрещено. В заключение отметим, что утверждения Пролога могут располагаться пользователем и на одной строке, а сама система в дальнейшем произведет их перераспределение по строкам.

Ниже приведен текст программы на Прологе в стандартной форме, полученный в результате выполнения команды `LIST ALL` (не следует путать с текстом, полученным после выполнения команды `list all` в системе SIMPLE):

```
((owns Jim Spot)
 (white Spot))
((owns Joe X)
 (isa X dog)
 (NOT white X))
((isa Tom man))
((isa X black-dog)
 (isa X dog)
 (black X))
((isa X animal)
 (OR ((isa X dog)) ((isa X cat))))
((infix isa))
((infix owns))
((postfix black))
((postfix white))
((dict isa))
((dict owns))
&.
```


Такой текст несколько труднее для восприятия, чем текст, производимый системой SIMPLE. Но следует отметить, что по мере накопления опыта большинство пользователей предпочтет все-таки стандартный синтаксис. Это произойдет прежде всего потому, что система SIMPLE занимает объем оперативной памяти, равный 16К, а такого объема памяти вполне достаточно для размещения большой программы на Прологе.

2.5. МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ

Преимущество системы программирования микроПролог заключается в ее модульности. Например, программа SIMPLE, входящая в систему программирования микроПролог, состоит из трех модулей: `program-mod`, `errmsgs-mod`, `query-mod` *. Первый модуль предназначен для ввода программы, второй — для проведения синтаксического анализа и выдачи сообщений об ошибках и третий — для обработки запроса программой, которая была сконструирована. Эти модули занимают объемы 1К, 9К и 6К оперативной памяти соответственно. После того, как разработка программы полностью закончена, модуль `program-mod` можно удалить из оперативной памяти, используя для этого команду `KILL program-mod`. Освобожденное пространство можно использовать для обработки запросов. Ничто не может повергнуть в большую ярость, чем такая ситуация, когда программист, затратив много времени на разработку программы, в ответ на первый запрос получает сообщение: «нет свободного пространства в оперативной памяти». Удаление модуля `program-mod` — один из возможных путей решения этой проблемы. Кроме того, пользователи могут прийти к заключению, что в одном или нескольких модулях существуют такие средства, которые им никогда не понадобятся. В этом случае разрешается разрабатывать свои собственные модули, реализующие как имеющиеся в системе, так и дополнительные возможности. Это позволит создавать эффективные системы, ориентированные на решения конкретных задач. Подробное описание всех системных модулей читатель может найти в справочном руководстве по микроПрологу.

2.6. ИНФОРМАЦИЯ ДЛЯ ПОЛЬЗОВАТЕЛЕЙ ЭВМ SPECTRUM

Версия микроПролога, реализованная на ЭВМ Spectrum, содержит около 50 программ. В зависимости от выполняемых функций эти программы делятся на группы. Их полное описание дано в справочном руководстве. Здесь же хотелось обратить внимание на ряд команд, касающихся операций ввода-вывода

* Program — программа, errmsgs — сокращение от error messages — сообщения об ошибках, query — запрос. — *Прим. ред.*

на терминал. Использование этих команд позволит пользователям легко управлять работой ЭВМ.

CLS N. Эта команда очищает экран, помещает в левый верхний угол символ приглашения & и окрашивает экран в цвет с кодом N, где N — число в интервале от 0 до 7.

CLOSE (нмя файла). Необходимо выполнять эту команду перед любой попыткой загрузить файл или сохранить файл в случае появления ошибки. Использование этой команды позволяет надеяться, что пользователь запомнит точное нмя содержащего его программу файла, которую он хочет либо загрузить в оперативную память, либо сохранить.

BREAK (Остановка). Чтобы реализовать эту команду, необходимо нажать две клавиши — **SYMBOL SHIFT** и **SPACE**. Результатом является остановка выполнения программы. После этого на терминал будет выдано сообщение об ошибке, но его следует игнорировать.

SCROLLING (Просмотр). Для приостановки просматривания (лнстания) длинной программы используйте клавишу **STOP** (или клавиши **SYMBOL SHIFT** и **A**). Чтобы возобновить просмотр, достаточно нажать те же самые клавиши. Фактически эта команда инициирует прерывание, которое временно задерживает выполнение любой программы.

IN/OUT (ввод-вывод). Команда

? ((PIO M N))

осуществляет пересылку значения N в порт с номером M. Команда

? ((PIO M X) (PP X))

считывает значение из порта M и выводит его на экран дисплея *.

Предоставляемая память. Команда

? ((SPACE X) (PP X))

печатает объем оставшейся в распоряжении пользователя памяти в киллобайтах. Кроме того, выполнение этой команды приводит к очищению памяти от данных, оставшихся от обработки предыдущих запросов **.

2.7. ЗАПРОСЫ

Извлекать информацию из программы можно с помощью различных по форме запросов. Некоторые из них будут показаны ниже и адресованы следующей короткой программе:

* Наряду с использованием словосочетания «выводит на экран дисплея» в дальнейшем будет также использован термин «печатает». — *Прим. пер.*

** Отметим, что две последние команды удовлетворяют требованиям стандартного синтаксиса языка Пролог и в связи с этим могут быть использованы в программах, написанных на этом языке. — *Прим. пер.*

Flash isa dog	; Флэш это собака
Rover isa dog	; Ровер это собака
Bootsie isa cat	; Бутси это кошка
Star isa horse	; Стар это лошадь
Flash black	; Флэш черная
Bootsie brown	; Бутси коричневая
Rover red	; Ровер рыжая
Star white	; Стар белая
X house-trained if	; X домашнее-животное, если
(either X isa dog	; (либо X это собака или X
or X isa cat)	это кошка)
X isa animal if	; X это животное, если
(either X isa horse	; (либо X это лошадь
or X house-trained)	; или X домашнее-животное)
Tom owns X if	; Том владеет X, если
X isa dog and	; X это собака и
not X black	; X не черного-цвета
Kate owns X if	; Кейт владеет X, если
(either X black	; (либо X черного-цвета или X это
or X isa horse)	лошадь)

Во-первых, можно проверить, имеется ли в программе тот или иной факт. Например, на запросе

is (Rover red)
[верно (Ровер рыжая)]

должен быть получен ответ

YES
[ДА]

В более общем случае можно проверить, существуют ли значения аргументов, делающих отношение истинным:

is (x isa dog)
is (Star isa x)
is (Spot isa x)

На первый запрос будет получен ответ YES, поскольку X может принять значение, делающее отношение истинным. Второй и третий запросы позволяют установить, существуют ли в программе отношения «isa», в которых Star и Spot являются первыми аргументами. Результатами обработки этих запросов являются YES и NO соответственно.

Приведем еще несколько примеров запросов

all (x: x isa dog) *
all (x: Kate owns x)
all (x: x owns y)
all (x y: x owns y)
all (x y z: x owns y and y isa z)

* Определять все (x: x это собака). — Прим. пер.

С их помощью можно определить клички всех собак; клички всех животных, которыми владеет Кейт; имена всех тех людей, которые кем-нибудь владеют; то же самое, что и в предыдущем запросе, и, кроме того, клички всех животных, которыми кто-нибудь владеет; то же самое, что и в предыдущем запросе, но еще и виды животных.

Найти только одно значение аргумента можно, например, так:

```
one (x: x house-trained)
[определить одно (x: x домашнее животное)]
Flash
[Флэш]
more? (y/n) y
[еще? (да/нет) да]
Rover
[Ровер]
more? (y/n) n
[еще? (да/нет) нет]
&.
```

Если требуется дополнительно найти другое значение аргумента, пользователь должен ответить *y* (да).

Заметим, что особенно внимательным надо быть при использовании в предложениях и запросах оператора «not». В предложении

Tom owns x if x isa dog and not x black

условие состоит из двух частей, причем обратите внимание на то, что оператор «not» содержится во второй. Можно попробовать сформулировать предложение так:

Tom owns x if not x black

стараясь определить то, что Том является владельцем всех животных нечерного цвета. Но таким путем это делать нельзя. Если такое предложение все же есть, то Том не будет считаться владельцем кого бы то ни было. Аналогично обстоит дело с запросами.

Рассмотрим запрос

all (x: not x isa dog)

На него система всегда будет отвечать NO. Но, например, на запрос

all (x: x isa animal and not x isa dog)

будут получены два ответа: Star (Стар) и Bootsie (Бутси).

Упражнение 2.1

Составьте запросы, позволяющие определить:

- а) всех, кто владеет животными;
- б) всех, кто владеет животными небелого цвета;

- в) того, кто владеет Бутси;
г) клички тех животных, которыми кто-то владеет, и имена владельцев.

Упражнение 2.2

Что будет получено в ответ на следующие запросы:

- а) is not Rover red?
б) all (x: owns x and not x black);
в) all (x: x owns y and not y black).

Упражнение 2.3

Напишите предложение, описывающее животных светлого (нечерного) цвета, а затем составьте запрос, позволяющий определить всех тех нечерных животных, которые не являются лошадьми.

Логическая дедукция используется во многих областях искусственного интеллекта. Рассмотрим запрос: определить, какие животные не являются домашними. Поскольку этот запрос не очень сложен, человек способен ответить на него, просто просмотрев текст программы. При решении реальных задач возникают ситуации, когда человеческих возможностей может не хватить. Тогда приходится прибегать либо к помощи карандаша и бумаги, либо к помощи ЭВМ. Отметим те факторы, которые делают извлечение необходимой информации затруднительным:

огромное число рассматриваемых данных;

большое число не относящихся к решаемой задаче данных;

такое большое число различных путей, возникающих в процессе решения задачи, что обычных человеческих способностей недостаточно, чтобы ориентироваться в них.

Попробуйте, не пользуясь никакими дополнительными инструментами, кроме своей головы, решить следующую задачу. Бутси — коричневая кошка, Корни — черная кошка и Мактэвити — рыжая кошка. Флэш, Ровер и Спот — собаки; Ровер — рыжая, а Спот — белая. Все животные, которыми владеют Том и Кейт, имеют родословные. Том владеет всеми черными и коричневыми животными, а Кейт владеет всеми собаками небелого цвета, которые не являются собственностью Тома. Алан владеет Мактэвити, если Кейт не владеет Бутси и если Спот не имеет родословной. Флэш — пятнистая собака. Определить, какие животные не имеют хозяев?

Использование программы, листинг которой приведен ниже, позволяет немедленно получить ответ на запрос all (x: x is a animal and not y owns x):

Flash brindled
 Flash isa dog
 Rover isa dog
 Spot isa dog
 Korky isa cat
 Bootsie isa cat
 McTavity isa cat
 X isa animal if
 (either X isa dog or X isa cat)
 Spot white
 Rover red
 Korky black
 Bootsie brown
 McTavity ginger
 Tom owns X if
 X isa animal and
 (either X black or X brown)
 Kate owns X if
 X isa dog and
 not X white and
 not Tom owns X
 Alan owns McTavity if
 McTavity isa cat and
 not Kate owns Bootsie and
 not Spot has-pedigree
 X has-pedigree if
 (either Tom owns X or Kate owns X)

Если читатель все же справился с этой работой сам, он должен получить ответ «Спот». Методы принятия решений, используемые даже в такой простой задаче, как эта, распространяются на такие области искусственного интеллекта, как игры, решение задач, доказательство теорем, а также проектирование и эксплуатацию баз данных.

2.8. СРЕДСТВА, ПОЗВОЛЯЮЩИЕ РАБОТАТЬ С ЧИСЛОВОЙ ИНФОРМАЦИЕЙ

Следует предупредить, что микроПролог специально не ориентирован на работу с числовой информацией. У него другие области применения. Однако ряд имеющихся в Прологе встроенных арифметических функций чрезвычайно полезен практически во всех приложениях. Для вычисления значений арифметических выражений предназначен специализированный модуль EXPTRAN. В случае необходимости этот модуль должен быть загружен в оперативную память в дополнение к другим используемым в данный момент модулям. Существует также ряд включенных в стандартный синтаксис языка арифметических операций, которые можно использовать без подзагрузки дополнительных модулей. К ним относятся SUM, TIMES, LESS, INT и SIGN. Все они вместе с другими

отношениями и командами могут быть использованы в программах, написанных на микроПрологе. Ниже дано описание этих арифметических операций, представленных в форме отношений.

SUM. Это отношение имеет вид $SUM(X\ Y\ Z)$. Его можно интерпретировать как высказывание, которое истинно (принимает значение TRUE), если сумма X и Y равна Z , и ложно (FALSE) — во всех остальных случаях. Допускается использование целых чисел и чисел с плавающей запятой.

Например, отношения $SUM(4\ 5\ 9)$ и $SUM(0.1-1.2-1.1)$ истинны, а отношения $SUM(5\ 7\ 22)$ и $SUM(7.2\ 5.2\ 6.8)$ ложны. Покажем, как можно использовать отношение SUM.

1. *Для проверки сумм в правиле программы или запросе*
Пусть имеется правило

$X\ \text{valid if } SUM(25\ 5\ Y)$

В ответ на запрос $is\ X\ \text{valid}$ будет получено YES в случае, если $Y = 30$, и значение NO во всех остальных случаях.

2. *Для сложения двух чисел*

В результате выполнения $SUM(3\ 5\ X)$ переменной X будет присвоено значение 8, а в результате вычисления $SUM(9 - 2\ X)$ — значение 7.

3. *Для определения разности двух чисел*

В результате вычисления $SUM(X\ 3\ 7)$ X получает значение 4, а вычисление $SUM(3\ X - 5)$ приведет к тому, что X будет равно -8.

TIMES. Это отношение можно использовать для умножения двух чисел, деления двух чисел и для проверки результата, получаемого при умножении двух чисел. Отношение представляется в виде

$TIMES(X\ Y\ Z)$

и принимает значение TRUE, если произведение X и Y равно Z и FALSE — во всех остальных случаях.

1. *Проверка произведения*

В результате проверки выражения $TIMES(X\ 2\ 6)$ будет получен ответ TRUE, если X равен 3.

$TIMES(4\ 7\ X)$ дает TRUE, если X равен 28.

2. *Умножение*

В результате вычисления $TIMES(4\ 5\ X)$ X получит значение 20.

3. *Деление*

В результате вычисления $TIMES(X\ 6\ 42)$ X получит значение 7. Вычисление $TIMES(3\ X\ 39)$ приводит к тому, что X примет значение 13.

LESS. В этом отношении используются два аргумента. $LESS(X\ Y)$ примет значение TRUE, если X строго меньше, чем Y . Например, вычисление $LESS(5\ 6)$ даст TRUE, в то время как вычисление $LESS(5\ 5)$ — FALSE. Аргументами этого отношения

также могут быть символы. Тогда сравниваются их коды. Например, ответом на запросы

is LESS (A B) и is LESS (AY AZ)

будет YES, а на запросы

is LESS (AD AC) и is LESS (& %)

будет NO.

INT. Это отношение используется для определения ближайшего к данному числу целого. Из двух возможных результатов округления выбирается тот, который ближе к нулю. Например, вычисление

which (x: INT (3.6 X))

приносит 3, а вычисление

which (x: INT (-3.9 X))

дает -3.

Первым аргументом отношения должно всегда быть либо число, либо выражение, результатом вычисления которого является число. Отношение INT можно использовать и для того, чтобы проверить, является ли заданное число целым.

Например, ответом на запрос

is (5 INT)

будет YES.

SIGN. Отношение имеет вид SIGN (X Y). В результате вычисления этого отношения Y будет присвоено одно из значений — 1, 0 или —1 в зависимости от того, положительно, равно 0 или отрицательно X. Первым аргументом этого отношения всегда должно быть конкретное число. В результате вычисления SIGN (-2.5 Y) Y присваивается значение, равное -1. В результате выполнения SIGN (2.5 Y) Y будет равно 1.

Упражнение 2.4

Определить ответы, которые будет давать система Пролог на следующие запросы:

- | | |
|----------------------------|-------------------------------|
| а) all (x: SUM (4 5 x)); | б) all (x: SUM (x — 1 2)); |
| в) all (x: SUM (5 X 2)); | г) all (x: TIMES (3 6 x)); |
| д) all (x: TIMES (4 x 8)); | е) is (SUM (5 6 7)); |
| ж) is (SUM (3 — 8 X)); | з) объяснить последний ответ. |

Упражнение 2.5

Определить ответы на следующие запросы:

- | | |
|---|---------------------------|
| а) all (x: SIGN (2 x)); | б) all (x: INT (-4.9 x)); |
| в) all (x: INT (x1)); | г) all (x: SIGN (x 0)); |
| д) объяснить ответы на запросы (в) и (г). | |

2.9. ОТНОШЕНИЯ, ЗАДАВАЕМЫЕ ПОЛЬЗОВАТЕЛЕМ

В Прологе разрешено использовать встроенные отношения для конструирования других полностью определяемых пользователем отношений. Например, можно задать отношение «меньше или равно» следующим образом:

```
x leq x
x leq y
if x LESS y
```

Первая строка этого определения используется для констатации того факта, что любое число меньше или равно самому себе. Во второй строке утверждается, что x меньше или равно y , если x строго меньше y . В запросах это отношение можно использовать так:

```
is (2 leq 5)
is (2 leq 2)
```

Ответом на оба эти запроса будет YES.

Упражнение 2.6

Определите отношение «больше или равно», назвав его `geq`.

Упражнение 2.7

Прокомментируйте следующие запросы:

а) `is (2 leq x)`; б) `is (x leq 2)`; в) `all (x: x leq 5)`.

Запросы, аналогичные `is (2 leq 5)`, не имеют особого смысла, поскольку ответ известен заранее. Но следует отметить, что некоторые очень важные отношения можно определить, используя отношение `leq`. Предположим, необходимо сформулировать отношение `in`, определяющее числа, во-первых, принадлежащие заданному интервалу и, во-вторых, отстоящие друг от друга на одно и то же значение, и использовать его для генерации и проверки правильности этих чисел.

Первая предварительная попытка может выглядеть так:

```
x in (y z) if
  y leq x and
  x leq z
```

Здесь обозначение `(y z)` использовано для того, чтобы показать, что нижней границей интервала является y , а верхней — z . Интервал, в который включены как нижняя, так и верхняя границы, называется отрезком или закрытым интервалом. Открытым называется интервал, которому не принадлежат либо обе его границы, либо одна из них.

Рассмотрим запросы

is (5 in (2 9))
is (3 in (6 9))

Оба эти запроса правильны. Ответами на них являются YES и NO соответственно. Таким образом, определенное выше отношение можно использовать для проверки принадлежности заданного числа заданному интервалу. Но обработка запросов вида

all (x: x in (4 6))

приводит к ошибкам. Попробуем объяснить, почему. Объяснение надо искать в анализе последних двух заданий упражнения 2.5. Наш пример приводит к необходимости сначала выполнить операцию LESS (4 x). Но ее выполнить невозможно, поскольку значение x неизвестно.

Фактически заданному интервалу принадлежит очень много чисел: 4.0, 4.1, 4.23, 4.2345 и т. д., т. е. для того чтобы четко определить числа, включаемые в интервал, необходимо задать еще и разность между двумя любыми последовательными значениями, принадлежащими интервалу. Принимая это во внимание, попробуем записать программу так:

```
X in (X Y) if
  X leq Y
X in (Y Z) if
  SUM(Y 1 x) and
  x leq Z and
  X in (x Z)
```

Обратите внимание на использование в этой программе больших и малых букв.

Первое правило программы гарантирует, что нижняя граница всегда будет меньше верхней, и делает нижнюю границу первым искомым числом. Второе правило представляет собой механизм генерации остальных чисел, входящих в интервал. Этот механизм обеспечивает добавление 1 к нижней границе, проверяет, меньше или равен результат верхней границе, и, наконец, устанавливает, принадлежит ли X только что определенному интервалу. Таким образом будет формироваться возрастающая последовательность чисел с шагом, равным единице.

Например, в ответ на запрос

all (x: x in (3 7))

будут получены все целые в интервале от 3 до 7.
Аналогично, в ответ на запрос

all (x: x in (2.5 5.5))

будет получена следующая последовательность чисел:

2.5, 3.5, 4.5, 5.5.

Для того чтобы установить любой нужный Вам шаг, необходимо изменить отношение SUM во втором правиле следующим образом:

SUM (Y n x)

где n — желаемое значение шага. Тогда для генерации последовательности с шагом, например, 0.5 необходимо использовать

SUM (Y 0.5 x)

Заметим, что во втором правиле отношение in определяется через самое себя. Такие определения называются рекурсивными. Рекурсия допускается в таких языках программирования высокого уровня, как Алгол и Паскаль. В этих языках использование рекурсии сводится к вызову процедур самих себя. В Прологе использование рекурсии позволяет добиться ясности и в то же время лаконичности при конструировании правил. Рекурсия довольно удобна для задания числовых отношений, но может быть использована и в других случаях, например, для обработки списков. Но этим вопросом мы займемся позднее.

Упражнение 2.8

Используя отношение leq, определить отношение «больше чем», обозначив его gt.

Упражнение 2.9

а. Используя приведенное выше отношение in, определить, что будет напечатано в ответ на запрос

all (x: x in (0 n))

б. Сформировать запрос для получения всех целых чисел в диапазоне от 1 до 100.

в. Переписать второе правило в определении отношения in таким образом, чтобы обеспечить генерацию последовательности 0, 0.1, 0.2 и т. д.

г. Модифицировать отношение in таким образом, чтобы обеспечить получение чисел в интервале (X Y), не включая в него верхней границы Y.

д. Модифицировать определение отношения in для получения убывающей последовательности чисел.

Отношение in можно использовать по-разному. Например, с его помощью можно определить все делители заданного числа. Это может быть сделано так:

all (x y: x in (2 28) and y in (2 28) and TIMES (x y 28))

В ответ на такой запрос будет напечатано

2	14
4	7
7	1
14	2

Для того чтобы устранить повторения, перепишем запрос иначе:

```
all (x y: x in (2 28) and y in (2 28) and x LESS y and TIMES (x
y 28))
```

Можно оформить процесс получения делителей в виде отдельного отношения:

```
(X Y) factor Z if
  X in (2 Z) and
  Y in (2 Z) and
  X LESS Y and
  TIMES (X Y Z)
```

2.10. ОТНОШЕНИЯ, СЛУЖАЩИЕ ДЛЯ ВВОДА ДАННЫХ

Знакомые с Бейсиком наверняка знают предназначенные для ввода данных операторы этого языка, такие, как INPUT X или INPUT «ваше имя»; NS. С их помощью программа запрашивает у пользователя значение арифметической или символьной переменной. Полностью аналогичных средств в Прологе, естественно нет. Но в нем предусмотрено специальное отношение is-told, позволяющее запрашивать у пользователей ту или иную информацию. Это отношение расположено в файле TOLD, который можно загрузить в оперативную память с помощью команды LOAD TOLD (все это справедливо для микроПролога ЭВМ Spectrum). Модуль, реализующий функции этого отношения, называется told-mod и занимает 1К.

Предположим, что в программе присутствует следующее отношение:

```
X known if
  (x name) is-told.
```

Следующий диалог пользователя и ЭВМ показывает, как такое отношение можно использовать:

& all (x: x known)	; Запрос пользователя
{определить все (x: x известно)}	
X name ? ans Tom	; Вопрос системы и ответ пользователя
{имя X ? ответ Том}	
Tom	; Ответ системы
{Том}	
X name ? just Alan	; Вопрос системы и ответ пользователя
{имя X ? последний ответ Алан}	
Alan	; Ответ системы
{Алан}	
No (more) answers	; Конец диалога
{Ответов (больше) нет}	

Если данный ответ пользователя является последним, то он должен сопроводить его словом just. Кроме того, пользователь может ответить на вопрос просто no.

Другим способом использования отношения is-told являются запросы, порождающие вопросы, на которые пользователь должен ответить «да» или «нет». Например,

```
& is (Sam known)
Sam name ? yes
YES
& is (Ted known)
Ted name ? no
NO
&.
```

В предыдущем примере система лишь повторяла сообщения пользователя, которые, впрочем, тоже имели небольшое практическое значение, т. е. предыдущий пример использовался только для демонстрации читателям правил работы с отношением is-told. Но на практике это отношение весьма полезно особенно в двух следующих случаях:

is-told может быть использовано при определении другого отношения; тогда Пролог-система будет запрашивать данные у пользователя для того, чтобы проверить какой-нибудь факт или получить информацию;

is-told может быть использовано для включения новых фактов в базу данных; другими словами, Пролог-система будет добавлять новые строки в программу пользователя.

Кроме того, допустимо уничтожение строк, т. е. утверждений программы. Так что можно создавать программы, которые в процессе работы будут изменять самих себя, реагируя таким образом на сообщения пользователей.

Рассмотрим программу, анализ которой позволит понять, как is-told используется для определения отношения older:

```
Alan aged 15
Bill aged 42
Colin aged 28
X older Y if
    X aged Z and
    (Y aged x)is-told and
    x LESS Z
```

Ниже приведен диалог между ЭВМ, выполняющей эту программу, и пользователем:

```
all (Colin is older than x: Colin older x)
[определить все (Колин старше чем x: Колин старше x)]
X aged Y ? ans Dave 20
[X имеет возраст Y? ответ Дейв 20]
Colin is older than Dave
[Колин старше чем Дейв]
```

X aged Y ? ans Eric 44
 [X имеет возраст Y? ответ Эрик 44]
 X aged Y ? just Fred 22
 [X имеет возраст Y? только Фред 22]
 Colin is older than Fred
 [Коллин старше чем Фред]
 No (more) answers
 [Ответов (больше) нет]

При обмене вопросами и ответами Пролог-система просит пользователя сообщить ей чье-нибудь имя и возраст: X aged Y ? Если Colin старше, то печатается соответствующее сообщение. Затем запрашиваются новое имя и возраст. Так продолжается до тех пор, пока пользователь не захочет прекратить диалог, введя по нли just.

Можно было бы сформулировать запрос так:

all (x: Colin older x)

Тогда печатались бы только введенные пользователем имена тех, кто моложе, чем Коллин. В связи с этим заметим, что любые константы, такие, как «Colin is older than», можно вводить в текст запроса. Такого рода константы придадут диалогу между ЭВМ и пользователем более естественную форму. Никакого влияния на процесс получения ответа на запрос они не оказывают.

Программы, аналогичные приведенной выше, позволяют сопоставлять информацию, взятую из базы данных, с введенными пользователем данными.

Отношение is-told используется также для того, чтобы дать возможность пользователям вносить новые факты в базу данных, одновременно проводя сравнение этих новых фактов с уже существующими.

Ниже приведена новая версия отношения older.

X older Y if
X aged Z and
(Y aged x) is-told and
x LESS Z and
(Y aged x) add

Терм (Y aged x) add позволяет вносить в базу данных новую информацию, полученную системой от пользователя. Например, в результате диалога:

all (Colin is older than x: Colin older x)
 [определить все (Коллин старше чем x: Коллин старше x)]
 X aged Y ? ans Том 11
 [X имеет возраст Y ? ответ Том 11]
 Colin is older than Tom

[Колин старше чем Том]
 X aged Y ? ans Sam 42
 [X имеет возраст Y ? ответ Сэм 42]
 X aged Y ? no
 [X имеет возраст Y ? нет]
 No (more) answers
 [Ответов (больше) нет]

в базу данных будет добавлена только информация о возрасте Тома. Попробуйте объяснить, почему данные о возрасте Сэма в базу не попадут?

Упражнение 2.10

а. Объясните, по какой причине данные о людях старше Колина, полученные в ответ на вопросы, выдаваемые системой пользователю, в базу данных на заносятся?

б. Измените правило, описывающее отношение older так, чтобы любые данные о возрасте того или иного человека записывались в базу данных независимо от того, старше он кого-нибудь или нет. Выполните упражнение 2.10 перед тем, как работать с материалом книги дальше.

В определении отношения older условие $x \text{ LESS } Z$ предшествует терму $(Y \text{ aged } x) \text{ add}$. Если условие $x \text{ LESS } Z$ ложно, то программа считает текущую попытку интерпретации неудачной и завершает ее. Поэтому действия, связанные с $(Y \text{ aged } x) \text{ add}$, не выполняются.

Для того чтобы новая информация всегда заносилась в базу данных, необходимо два последних термина отношения older поменять местами. Тогда отношение будет иметь вид

X older Y if
 (Y aged x) is-told and
 (Y aged x) add and
 X LESS Z

Строки можно удалять из отношения так же легко, как и добавлять их. Как известно, модификация информации сводится к добавлению новых данных и удалению старых.

Следующая программа позволяет модифицировать саму себя:

Alan aged 15
 Bill aged 42
 Colin aged 28
 X older Y if
 X aged Z and
 (Y aged x) is-told and
 (Y aged x) add and
 (Y aged y) delete and
 x LESS Z

Ниже приведен протокол ее работы:

all (Colin is older than x: Colin older x)
[(Коллин старше чем x: Коллин старше x)]
X aged Y ? ans Alan 16
[X имеет возраст Y ? ответ Алан 16]
Colin is older than Alan
[Коллин старше чем Алан]
No sentence (Alan aged X)
[Удаление предложения (Алан имеет возраст X)]
Colin is older than Alan
[Коллин старше чем Алан]
X aged Y ? just Bill 43
[X имеет возраст Y ? последний ответ Билл 43]
No sentence (Bill aged X)
[Удаление предложения (Билл имеет возраст X)]
No more answers
[Ответов (больше) нет]

Если теперь посмотреть на программу, то легко заметить, что возраст Алана и Билла изменился. Отметим, что любое имя, вводимое пользователем в ответ на запрос системы вида X aged Y? должно присутствовать в базе данных. Программа, анализируя терм Y aged x, согласует Y с введенным именем, а x с введенным возрастом. После этого происходит сравнение введенного пользователем возраста с возрастом, хранимым в базе данных.

Выполнение конструкции (Y aged y) delete приводит к реальным действиям, только если в базе существуют данные о возрасте человека, имя которого введено пользователем. Эти действия заключаются в удалении из базы данных предложения, которое связывает это имя с новым возрастом, отличным от возраста, введенного пользователем.

Очень важно, говоря об отношении is-told, помнить, что информация, вводимая пользователем в ответ на вопрос системы, не запоминается, т. е. не заносится в базу данных. Эта информация используется только во время обработки текущего запроса. В приведенных выше запросах речь шла о конкретном человеке — Коллине. Соблазнительно попробовать определить с помощью запроса

all (xy: x older y)

имена всех тех, кто старше человека, имя и возраст которого будут введены пользователем. Но добиться этого с помощью отношения older нельзя. Покажем, почему это так. Для того чтобы упростить пример, удалим из отношения older термы, связанные с помещением в базу данных и исключением из нее утверждений. Теперь можно сравнивать возраст людей, информация о которых имеется в базе данных, с возрастом тех, кого, возможно, и нет в базе данных и информация о которых вводится пользователем. Диалог между ЭВМ и пользователем в этом случае может выглядеть так:

all (x is older than y: x older y)
 [определить все (x старше чем y: x старше y)]
 X aged Y ? ans Tom 22
 [X имеет возраст Y ? ответ Том 22]
 X aged Y ? just Ted 12
 [X имеет возраст Y ? последний ответ Тед 12]
 Alan is older than Ted
 [Алан старше чем Тед]
 X aged Y ? just Sam 40
 [X имеет возраст Y ? последний ответ Сэм 40]
 Bill is older than Sam
 [Билл старше чем Сэм]
 X aged Y ? ans Joe 27
 [X имеет возраст Y ? ответ Джо 27]
 Colin is older than Joe
 [Колин старше чем Джо]
 X aged Y ? just Fred 30
 [X имеет возраст Y ? последний ответ Фред 30]
 No (more) answers
 [Ответов (больше) нет]

Обратите внимание на то, что в ответ на сообщение о возрасте Тома (ans Том 22) никакой информации получено не было, хотя известно, что и Колин, и Билл старше Тома. Это произошло потому, что Пролог-система не сравнивала возраст Тома с возрастом Колина и Билла. Сравнение производилось только с возрастом Алана. И так будет продолжаться до тех пор, пока пользователь в ответ на вопрос системы не введет либо just, либо no. После этого, получив ответ

Alan is older than Ted

Пролог-система будет сравнивать все последующие вводимые пользователями данные с данными, относящимися к Биллу. И так до тех пор, пока вновь не будет введено just или no. Тогда Пролог-система перейдет к Колину. Все это очень хорошо только тогда, когда есть уверенность, что пользователь точно знает, с каким утверждением он работает. Но, естественно, пользователь в любую минуту может об этом забыть.

Для того чтобы пользователь мог определить, с каким утверждением он хочет работать, введен еще один вариант термина is-told. После введения этого нового термина в программу она примет вид

Alan aged 15
 Bill aged 42
 Colin aged 28
 X older Y if
 X aged Z and
 (Compare with X) is-told and
 (Y aged x) is-told and
 x LESS Z

Ниже приведен один из возможных вариантов диалога между программой и пользователем:

all (x is older than y: x older y)
[определить все x старше чем y: x старше y]
Compare with Alan ? yes
[Сравнивать с Аланом ? да]
X aged Y ? ans Tom 4
[X имеет возраст Y ? ответ Том 4]
Alan is older than Tom
[Алан старше чем Том]
X aged Y ? ans Sam 20
[X имеет возраст Y ? ответ Сэм 20]
X aged Y ? just Ted 5
[X имеет возраст Y ? последний ответ Тед 5]
Alan is older than Ted
[Алан старше чем Тед]
Compare with Bill ? no
[Сравнивать с Биллом ? нет]
Compare with Colin ? yes
[Сравнивать с Колином ? да]
X aged Y ? ans Tom 4
[X имеет возраст Y ? ответ Том 4]
Colin is older than Tom
[Колин старше чем Том]
X aged Y ? just Fred 40
[X имеет возраст Y ? Фред 40]
No (more) answers
[Ответов (больше) нет]
&.

Отметим, что теперь программа требует от пользователя подтверждения, что именно с информацией о том человеке, имя которого ему только что сообщено, пользователь хочет работать. Если ответ пользователя yes, то система просит пользователя ввести новое имя и возраст. После этого она сравнивает только что введенный возраст с взятым из базы данных возрастом выбранного пользователем человека и, если первый меньше второго, выдает сообщение

NAME is older than name

где NAME — имя выбранного пользователем для сравнения человека; name — только что введенное имя.

Если же ответом является no или ответ начинается с just, то система, завершив обработку ответа, переходит к утверждению, описывающему следующего человека. Такая схема лучше, чем та, которая была раньше. Но все-таки не совсем удобно просматривать утверждения базы данных одно за другим до тех пор, пока требуемое не будет найдено. Было бы лучше, если бы пользователь мог

сразу же определить нужное утверждение и в случае необходимости тотчас прекращать работу, а не вводить для этого несколько раз по. К счастью, сделать это легко. Достаточно только поменять местами фразы, содержащие терм is-told. Тогда правило, описывающее отношение older, будет выглядеть так:

X older Y if
(Compare with X) is-told and
X aged Z and
(Y aged x) is-told and
x LESS Z

Упражнение 2.11

Проанализируйте работу программы, полученной в результате последней модификации. Используйте эту программу для сравнения сначала возраста Колина, а затем возраста Билла с возрастом Теда (25 лет), Джо (30 лет) и Тома (55 лет).

2.11. НА ПУТИ К ЕСТЕСТВЕННОМУ ЯЗЫКУ

Значительное внимание исследователи, работающие над проблемами искусственного интеллекта, уделяют вопросам облегчения взаимодействий человека с машиной. Достичь удобного для человека диалога можно, применяя световое перо, сенсорный экран или устройства типа «мышь», позволяющие быстро выбрать требуемую информацию из меню. Все эти средства чрезвычайно полезны; даже совершенно неподготовленные пользователи после непродолжительной тренировки могут их использовать. Кроме того, применение таких устройств позволяет освободить пользователей от трудоемких операций, связанных с набором на клавиатуре. Однако все это весьма далеко от использования для взаимодействия с ЭВМ обычной человеческой речи. Хотя в последнее время достигнут определенный прогресс в области анализа и распознавания речи, следует отметить, что существующие системы «понимают» лишь небольшое число слов.

Использование Пролога предоставляет программе более широкие возможности для организации диалога с ЭВМ на языке, близком к естественному. Так правильная с точки зрения естественного языка формулировка запроса приводит к удобному для восприятия человеком ответу машины. Например, запрос

all (x is older than y: x older y)

с этой точки зрения лучше запроса

all (x y: x older y)

Преимущество первого запроса становится особенно очевидным, если результаты работы программы предназначены для людей,

далеких от вычислительной техники. Предпочтительнее именно такие расширенные формы запросов. Но программисты могут добиться и большего, если будут более тщательно продумывать тексты отношений. Тем самым они освободят пользователей от необходимости придавать протоколам работы программ вид текстов на естественном языке.

Проиллюстрировать изложенное выше проще всего на примере отношения, содержащего предикат is-told, например older. Естественно, что для этого придется внести несколько изменений в правило, описывающее это отношение. После модификации отношение older будет выглядеть так:

X older Y if

(Compare with which person X) is-told and

X aged Z and

(Input a name Y and an age x for comparison) is-told and

x LESS Z

Диалог между пользователем и ЭВМ примет теперь более естественный вид:

all (x is older than y: x older y)

[определить все (x старше чем y: x старше y)]

Compare with which person X ? ans Alan

[С кем необходимо провести сравнение ? ответ Алан]

Input a name X and an age Y for comparison ? ans Ted 5

[Введите имя X и возраст Y для сравнения ? ответ Тед 5]

Alan is older than Ted

[Алан старше чем Тед]

Input a name X and a name Y for comparison ? just Tom 9

[Введите имя X и возраст Y для сравнения ? ответ Том 9]

Alan is older than Tom

[Алан старше чем Том]

Compare with which person X ? no

[С кем необходимо провести сравнение ? нет]

No (more) answers

[Ответов (больше) нет]

Отметим, что ответ пользователя just Том 9 заставляет программу перейти от одной формы запроса к другой. Не должно смущать то, что в программе для представления имени и возраста использованы переменные Y и x, в то время как при выводе запроса для тех же самых величин используются обозначения X и Y. Важно понять, что переменные в Прологе не имеют постоянных или глобальных значений. Наша программа сравнивает значения, сообщенные последователем, используя для этого механизм сопоставления образцов, который временно связывает с переменными эти значения.

Существует возможность сделать программу еще более удобной для пользователя. Например, можно освободить пользователя от необходимости вводить в запрос текст «x is older than y» (это делалось для того, чтобы этот текст появлялся в процессе диалога). Перед тем как модифицировать программу, рассмотрим стандартную форму запросов. Напомним, что до сих пор мы пользовались средствами диалоговой системы SIMPLE, которая позволяет составлять запросы на языке, близком к естественному. Первое, что необходимо отметить, заключается в том, что в стандартном синтаксисе Пролога не существует эквивалента для отношения all. Запрос на стандартном Паскале будет выглядеть так:

? ((older x y))

Символ «?» используется для обозначения запроса. После него следует сам запрос, заключенный в двойные скобки. Любой запрос, касающийся отношения и его аргументов, должен иметь тот же самый вид, который имеет отношение, записанное в соответствии с правилами стандартного синтаксиса. Для того чтобы вывести на экран стандартное описание отношения, достаточно использовать команду LIST. Например,

```
& LIST aged  
((aged Alan 15))  
((aged Bill 42))  
((aged Colin 28))
```

Теперь можно ввести запрос

? ((aged x y))

В ответ программа напечатает &. Это означает, что найдены такие значения x и y, которые содержатся в утверждении типа ((aged x y)), т. е. такой ответ полностью соответствует YES, получаемому в ответ на запрос в случае использования системы SIMPLE.

Рассмотрим еще один запрос

? ((aged x 90))

В ответ система напечатает знак вопроса и следом за ним символ &. Это означает, что ответ на запрос отрицателен, т. е. не существует такого x, который содержится в утверждении типа

((aged x 90))

Отметим, что знак вопроса в стандартном синтаксисе соответствует NO в случае использования SIMPLE.

Ни в первом, ни во втором случаях никакая дополнительная информация не появляется. Если же результат необходимо напечатать, следует использовать следующую конструкцию:

```
&? ((aged x y) (PP x y))  
Alan 15  
&.
```

Запрос (aged x y) позволяет определить искомые значения x и y и по команде (PP x y) эти найденные значения печатаются. PP означает красивая печать (pretty print). В данном случае использование команды PP позволяет после печати пары значений переходить на другую строку. Если бы использовалась команда P (print), все печаталось бы подряд на одной строке. Не существует команды, предписывающей программе найти все остальные значения x и y, удовлетворяющие отношению

(aged x y)

Если пользователям такая команда нужна, то они должны реализовать ее сами. Пользователям дается возможность определять командные отношения так же свободно, как и обычные отношения. Это позволяет создавать программы, ориентированные на решение задач конкретной проблемной области. В дальнейшем будет показано, как компактно записать текст отношения, удовлетворяющий требованиям стандартного синтаксиса. Но сейчас остановимся на использовании двух стандартных отношений (примитивов), описанных выше.

Напомним, что основной задачей считается создание таких отношений, использование которых освобождает пользователей от необходимости конструирования сложных запросов. Сложные запросы, близкие к предложениям естественного языка, создаются для того, чтобы ответы ЭВМ содержали максимально возможное количество информации в удобной для восприятия форме. Это особенно необходимо в тех случаях, когда протоколы работы программ распространяются среди заинтересованных лиц.

Предположим, в ЭВМ находятся три следующих утверждения:

Alan aged 15
Bill aged 42
Colin aged 28

Диалог между пользователем и ЭВМ вида

&all (x y: x aged y)
[определить все (x y: x имеет возраст y)]
Alan 15
[Алан 15]
Bill 42
[Билл 42]
Colin 28
[Колин 28]

может легко быть изменен на следующий:

&all (x is y years old: x aged y)
[определить все (возраст человека по имени x — y: x имеет возраст y)]
Alan is 15 years old

[Возраст человека по имени Алан — 15]
Bill is 42 years old
[Возраст человека по имени Билл — 42]
Colin is 28 years old
[Возраст человека по имени Коллин — 28]

Последний вариант протокола более удобен для восприятия. Кроме того, он позволяет однозначно интерпретировать выдаваемую информацию. Например, 15 может означать число детей Алана, или число его земельных владений, или что-либо подобное. Ведь очевидно, что сообщение

Alan 15

может трактоваться по-разному. Конечно, можно организовать строго форматированный вывод, где возраст, число детей и т. п. будут печататься в отдельных снабженных заголовками столбцах таблицы. Но такой способ не позволит добиться нужной гибкости и будет слишком расточительным в тех случаях, когда требуется лишь небольшое число данных, характеризующих ту или иную личность.

В данный момент наша задача заключается в том, чтобы освободить пользователей от необходимости формирования длинных запросов вида

all (x is y years old: x aged y)

Чтобы получить от ЭВМ удобные для восприятия сообщения достаточно использовать в каждом отношении команду печати. Например,

(Bill aged 42)

заменить на

(Bill aged 42 if ((PP Bill is 42 years old)) ?)

Заметим, что знак вопроса появляется теперь сразу после команды печати. Это наиболее легкий путь, позволяющий включать стандартные команды непосредственно в тексты утверждений, удовлетворяющих требованиям системы SIMPLE. Рассмотрим теперь, какова будет реакция Пролог-системы на один из возможных запросов

all (x y: x aged y)
[определить все (x y: x имеет возраст y)]
Alan 15
[Алан 15]
Bill is 42 years old
[Возраст человека по имени Билл — 42]
Bill 42
[Билл 42]
Colin 28

Видно, что информация, касающаяся отношений, которые не были изменены, печатается в том же виде, что и раньше; в то же время данные о Билле печатаются как в старом, так и в новом форматах. Это объясняется тем, что используемое в запросе отношение `all` предусматривает команду печати, и, кроме того, команда печати явно присутствует в утверждении, определяющем возраст Билла. Устранить ненужную печать можно, используя следующий запрос:

```
all (: x aged y)
[определить все (: x имеет возраст y)]
Bill is 42 years old
[Возраст человека по имени Билл — 42]
No (more) answers
[Ответов (больше) нет]
&.
```

Но все же, хотя сообщения типа

Alan 15

появляться теперь не будут, на месте каждого из них в протоколе будет оставлена пустая строка. Это происходит потому, что перед двоеточием, где обычно появляются имена переменных, предназначенных для вывода на печать, теперь ничего нет. Другими словами, в данном случае факт возникновения пустых строк говорит о том, что по команде `all` ничего не печатается.

Отметим несколько недостатков метода, основанного на включении команд печати во все утверждения программы. Во-первых, этот процесс потребует для большой программы значительного времени; во-вторых, будет занято много дополнительной памяти; в-третьих, новые термы могут создать дополнительные трудности для понимания некоторых достаточно длинных утверждений программы. Естественно, было бы лучше, если бы удалось добавить к каждому отношению одно специальное отношение, определяющее вывод нужной информации. На выполнение действий, связанных с обычными отношениями, эти новые отношения влиять не будут. Было бы удобно дать отношениям печати имена, похожие на имена тех отношений, для работы с которыми они предназначены. Например, для рассматриваемого отношения `aged` отношение, определяющее вывод на печать, можно назвать `age` или `print-age` или, наконец, `paged`. Автору больше нравится последний вариант. В общем любому отношению можно поставить в соответствие отношение печати, определяющее требуемую форму вывода информации. Теперь можно перейти к определению отношения `paged`:

```
X paged Y if
  X aged Y and
  ((PP X is Y years old)) ?
```


Работает это отношение так:

& all (: x paged y)

[определить все (: x печ.-имеет-возраст y)]

Alan is 15 years old

[Возраст человека по имени Алан — 15]

Bill is 42 years old

[Возраст человека по имени Билл — 42]

Colin is 28 years old

[Возраст человека по имени Колин — 28]

&.

Пустые строки между ответами появляются потому, что отношение all выполняет собственную команду печати, а форма запроса такова, что печатать ничего не надо. Если все же желательно избежать появления пустых строк, то можно заменить команду PP в отношении paged на команду P:

((P X is Y years old)) ?

Выполнение команды P не приводит к переводу строки. Ответы будут теперь печататься на следующих друг за другом строках, поскольку перевод строки выполнит команда PP, не явно присутствующая в all.

Упражнение 2.12

а. Объясните, что будет получено в ответ на запрос?

all (: x aged y)

б. Сформируйте отношение older, определяющее, когда x старше y, и отношение polder, обеспечивающее вывод информации из отношения older.

Упражнение 2.13

Определите отношения, ориентированные на хранение данных, которые приведены в таблице; обеспечьте красивую печать. Приведите примеры такой печати для различных характеристик личности, указанных в табл. 2.1.

Таблица 2.1

name (Имя)	age (Возраст), лет	birth-date (Дата рождения)	income (Годовой доход), долл.	house-number (Номер дома)	years-in (Число лет проживания в доме)	job (Профессия)	years-at (Стаж), лет
Alan	15	3-6-70	500	5	5	0	0
Bill	42	12-2-43	12 000	19	7	Ratcher (мясник)	23
Colin	28	4-9-57	9 850	3	2	teacher (учитель)	7
Diane	30	3-8-55	6 500	7	5	nurse (няня)	5
Eve	12	29-3-73	0	4	12	0	0
Frances	25	7-9-60	11 500	2	8	doctor (врач)	2

Упражнение 2.14

Определите отношение, которое будет выводить на печать всю информацию, относящуюся к данной личности, в следующей форме (на примере Колина):

«Колин, 28 лет, учитель, родился 4-9-57, годовой доход составляет 9850 фунтов стерлингов, стаж работы 7 лет, живет в доме номер 3 в течение 2 лет».

Для программы из упражнения 2.12 требуется немного памяти. Используя команду

`? ((SPACE X) (PP X))`

можно определить, сколько памяти остается незанятой; такой памяти в данном случае должно быть около 4К. Как уже отмечалось, команда `SPACE` осуществляет чистку памяти от данных, оставшихся в ней после обработки предыдущих запросов. Не надо забывать, что программа не должна занимать всю свободную память, поскольку определенная память нужна для обработки запросов. Если вы хотите получить в Ваше распоряжение дополнительный участок памяти, используйте команду `kill program-mod`. Но имейте в виду, что после этого все вновь вводимые отношения должны быть представлены в форме, удовлетворяющей требованиям стандартного синтаксиса. Кроме того, с помощью команды

`kill errmsg-mod`

может быть освобождено дополнительно 1К памяти. Правда, в этом случае вместо развернутых сообщений о синтаксических и других ошибках будут выдаваться только номера ошибок. Для того чтобы получить тексты диагностических сообщений, надо будет воспользоваться руководством.

Как Вы помните, в упражнении 2.14 требовалось придумать такое отношение, которое обеспечивало бы вывод всей информации, относящейся к данной личности. В Ваших интересах написать свою версию программы и тем самым приобрести определенный опыт программирования на Прологе.

Для тех, кто не может справиться с поставленной задачей, приводим возможный вариант программы:

```
X pstat Y if
  X age Z and
  X job x and
  not X job Ø and
  X born y and
  X earns z and
  X years-at X1 and
  X in Y1 and
  X years-in Z1 and
  ((P X, a Z years old x born on y earns z a year, has been a x
    for X1 years, and has lived at number Y1 for Z1 years)) ?
```

Диалог между пользователем и ЭВМ может иметь следующий вид:

one (: pstat x)
[определить один (: вывод-данных-об-общественном-положении x)]
Bill, a 42 years old butcher born on (12-2-43) earns 12 000 a year, has been a butcher for 23 years, and has lived at number 19 for 7 years
[Билл, 42 года, мясник, родился (12-2-43), годовой доход составляет 12 000 долларов, стаж работы 23 года, живет в доме номер 19 в течение 7 лет]
more? (y/n) no
[еще? (да/нет) нет]
& all (: Frances pstat x)
[определить все (: Фрэнсис вывод-данных-об-общественном-положении x)]
Frances, a 25 years old doctor born on (7-9-60) earns 11 500 a year, has been a doctor for 2 years, and has lived at number 2 for 8 years
[Фрэнсис, 25 лет, врач, родился (7-9-60), годовой доход составляет 11 500 долларов, стаж работы 2 года, живет в доме номер 2 в течение 8 лет]
No (more) answers
[Ответов (больше) нет]

Не забудьте при формировании отношения born заключить дату в скобки. В том случае, когда скобки отсутствуют, например:

Frances born 7-9-60

будет получено сообщение об ошибке. Дело в том, что Пролог-система будет интерпретировать 7-9-60 как два аргумента: 7- и 9-60 *. Однако можно представить отношение в виде

Frances born Sept-7-60

и ошибки в этом случае не будет.

В качестве имени отношения, обеспечивающего вывод, использовано pstat — сокращение от print status **. Можно, конечно, придумать любое другое имя, но лучше использовать имена, мнемонически отражающие суть выполняемых операций.

Отметим, что в этой главе не рассматривались программы, имеющие какое-либо практическое значение. Главной считалась

* В первом случае выражение начинается с цифры 7. В связи с этим 7 и все последующие цифры (если они есть) интерпретируются как элемент данных арифметического типа. Поскольку за цифрой 7 сразу же идет нецифровой символ, остальная часть выражения считается элементом данных символьного типа. — *Прим. ред. пер.*

** Вывод на печать данных, характеризующих общественное положение. — *Прим. ред. пер.*

задача изучения базовых понятий языка Пролог и демонстрация возможностей их применения для решения задач искусственного интеллекта. Автор надеется, что читатель, который одолел эту главу, разобрался в следующем:

1. Программа на языке Пролог представляет собой последовательность определяемых пользователями отношений. Каждое отношение состоит из имени отношения и из одного или нескольких аргументов.

2. В Пролог-системе имеются встроенные отношения. С их помощью пользователь составляет и редактирует программу, а также формирует запросы.

3. Встроенными являются также такие логические операции, как `and`, `or` и т. д. Пролог-система использует эти операции вместе с мощными и встроенными средствами сопоставления для обработки запросов последователей. Широкие возможности Пролога по сопоставлению запросов с утверждениями необходимы для работы, например, с отношением `pstat`.

4. Пролог позволяет легко справляться с обработкой различных типов данных (см. предыдущую программу).

5. Пролог обеспечивает достаточно большую гибкость при работе с базами данных.

6. Пролог дает возможность осуществлять ввод и вывод сообщений на естественном языке.

Ответы к упражнениям

Упражнение 2.1

- a) `all (x: y isa animal and x owns y)`
- б) `all (x: x owns y and y isa animal and not y white)`
- в) `all (x: x owns Bootsie)`
- г) `all (xy: x isa animal and y owns x)`

Упражнение 2.2

- a) NO
- б) Rover Star
- в) Tom Kate

Упражнение 2.3

`x light-coloured if`
`x isa animal and`
`not x black`
`all (x: x light-coloured and not x isa horse)`

Упражнение 2.4

a) 9; б) 3; в) —3; г) 18; д) 2; e) NO; ж) YES; з) задавая в качестве третьего аргумента `x`, мы тем самым спрашиваем, является ли сумма 3 и 6—числом.

Упражнение 2.5

a) 1; б) —4; в) ошибка в процессе вычисления; г) ошибка в процессе вычисления; д) ошибки в упражнениях (в) и (г) связаны с тем, что в отношениях `INT` и `SIGN` значение первого аргумента не определено.

Упражнение 2.6

x geq X
x geq y
if y LESS X

Упражнение 2.7

- а) YES, поскольку запрос в данном случае можно сформулировать так:
«2 меньше или равно какому-либо числу»;
б) YES, запрос трактуется так: «Какое-либо число меньше или равно 2»;
в) ошибка в процессе вычисления.

Упражнение 2.8

x gt y if ; x больше-или-равно y if
not x leq y ; not x меньше-или-равно y

Упражнение 2.9

- а) 0, 1, 2, 3, 4;
б) all (x: x in (1 100))
в) терм SUM должен принять вид SUM (Y 0.1 x)
г:

X in (X Y) if
X LESS Y
X in (Y Z) if
SUM (X 1 x) and
X I.ESS Z and
X in (x Z)

- д) необходимо поменять местами утверждения, определяющие отношение in

Упражнение 2.10

Смотрите следующее упражнение.

Упражнение 2.11

Модифицированное отношение позволяет пользователям с самого начала определять, с кем именно необходимо провести сравнение.

Упражнение 2.12

а. Должно быть выведено три пустые строки. Программа проведет обработку запроса, получит результаты, но ничего не напечатает, поскольку перед двоеточием в запросе ничего не стоит.

б:

X older Y if
X aged Z and
Y aged x and
x LESS Z
X polder Y if
X older Y and
((PX is older than Y)) ?

Упражнение 2.13

Ниже на Прологе описаны факты, касающиеся только человека по имени Колин. Для остальных эти утверждения имеют аналогичную структуру:

Colin age 28	
Colin born (4-9-57)	; обратите внимание на скобки
Colin earns 9850	
Colin in 3	; 3 — номер дома
Colin years-in 2	; 2 — число лет проживания в доме
Colin job teacher	
Colin years-at 7	; 7 — стаж работы

Отношения, обеспечивающие упорядоченный вывод данных, могут быть определены так:

- X page Y if**
X age Y and
((P X is Y years old)) ?
- X pborn Y if**
X born Y and
((P X was born on Y)) ?
- X pearn Y if**
X earns Y and
((P X earns \$ Y)) ?
- X pin Y if**
X in Z and
X years-in Y and
((P X has lived in number Z for Y years)) ?
- X pjob Y if**
X job Y and
not X job 0 and
((P X is employed as a Z)) ?
- X pat Y if**
X job Z and
X years-at Y and
((P X has been employed as a Z for Y years)) ?

Дадим некоторые пояснения. Точки с запятыми служат для того, чтобы разделить тексты отношений от комментариев. В каком-то смысле использование комментариев противоречит тому, что было сказано ранее о самодокументируемости программ, написанных на ориентированных на решение задач искусственного интеллекта языках. В принципе можно было бы выбрать имена отношений так, чтобы они отражали семантику отношений. Но все дело в том, что поскольку находящаяся в распоряжении пользователя оперативная память ЭВМ Spectrum ограничена, имена отношений должны быть короткими. В заключение отметим, что комментарии не обязательно вводить в ЭВМ.

Упражнение 2.14

См. материал, непосредственно следующий за упражнением 2.14.

3.1. СПИСОК КАК СТРУКТУРА ДАННЫХ

В конце второй главы была сделана попытка построения реляционной базы данных, содержащей информацию о нескольких людях. Для этой цели был использован целый ряд отношений. Первым аргументом каждого отношения было имя человека, а вторым — одно из характеризующих этого человека данных. Причем, число утверждений, задающих одно отношение, совпадало с числом различных людей. Отметим ряд недостатков, характерных для программ такого типа.

1. Такие программы быстро становятся очень длинными и трудными для сопровождения.

2. Необходимо чрезмерно большая работа по вводу фактов.

3. Программы не удовлетворяют требованиям гибкости и трудны для модификации. Каждый раз, когда вводится новый аргумент, необходимо модифицировать все отношения, которые его используют.

4. Такие программы занимают очень много места в памяти.

5. Достаточно трудно, а в определенных случаях и невозможно обнаружить и извлечь нужную порцию информации. В больших базах данных пользователям потребуется знать имена и определения всех отношений, предназначенных для работы с ними.

6. Принципы, положенные в основу построения программ, в большей степени определяют то, как программа будет использоваться. В частности, от них во многом будут зависеть используемые типы данных.

В предыдущей главе была сделана попытка устранить два последних недостатка. Для этой цели были разработаны специальные отношения, предназначенные для выдачи данных различных типов. Но все же необходимы дальнейшие усовершенствования.

Приступим теперь к изучению списков. Список — это такая структура данных, которая помогает сделать программу компактной, эффективной и легко управляемой. Причем последнее относится как к программистам, так и к пользователям. Списки — одна из тех структур, которые широко используются в программах искусственного интеллекта, и, кроме того, — это базовая структура языка Лисп. В Прологе также имеются мощные средства обработки списков, причем в их основе лежат методы, аналогичные тем, которые используются в Лиспе.

Идеи, лежащие в основе создания списка как структуры данных, используемой в вычислениях, берут свое начало в обычной жизни. Списки магазинов, списки книг для чтения, списки команд и инвентаризационные списки — все они состоят из нескольких элементов, имеющих, по крайней мере, одно общее свойство. В Прологе это свойство трактуется как отношение, определяющее связь между элементами в списке или между списком и элементом, внешним по отношению к списку. Ниже даны примеры списков.

1. (a b c d)
2. (a bc def ghi)
3. (1 2 3 4)
4. (Adam Eve Joe Tom)
5. (a2 Jim evergreen 25)
6. (menu roll rota portfolio)
7. ((a b c ()) (what is next ?) (This is a list) (o () zero null))

Каждый из этих списков содержит по четыре члена. Для некоторых списков связь между членами устанавливается легко, в то время как трудно, например, установить связь между элементами пятого списка. Шестой список состоит из элементов, являющихся названиями типов списков; седьмой — это список, состоящий из списков. В седьмом списке выделяется пустой список, обозначаемый (). Пустым называется список, не содержащий ни одного элемента. Отметим, что список (()) не является пустым: он содержит один член — пустой список. Понятие «список» имеет много общего с понятием «множество». Так же как и множество, список можно задать, либо указав все его элементы (например, кошки, собаки, крысы, мыши), либо определив правило формирования его членов (например, все положительные числа меньше 20).

При обработке списков часто используются два следующих отношения: APPEND и ON. Рассмотрим сначала отношение ON:

X isa list-type if

X ON (menu roll rota portfolio)

Диалог между пользователем и системой, основанной на использовании этого отношения, может выглядеть так:

all (x: x isa list-type).

[определить все (x: x это тип-списка)]

menu

[меню]

roll

[рулон]

rota

[расписание]

portfolio

[папка]

No (more) answers

[ответов (больше) нет]

Заметим, что числа, непосредственно предшествующие буквам, считаются самостоятельными термами, т. е. в ответ на запрос

all (x: x ON (a2 3b 33c 44d4 555))

будет напечатано a2, 3b, 33, c, 44, d4, 555.

В микроПрологе предусмотрен специальный оператор конструирования списков; он обозначается |. Выражение

X | Y

означает, что имеется конструкция, состоящая из элемента X, за которым следует список Y. Выражение

(XY | Z)

означает, что за элементами X и Y следует список Z. Другими словами, конструктор списков позволяет расчленить список (X | Y) на голову X и хвост Y.

Отношение APPEND служит для объединения двух списков. Например,

```
which (x: APPEND ((abc) (def) x))  
(abc def)  
No (more) answers
```

В рамках стандартного синтаксиса определение отношения ON будет выглядеть так:

```
((ON X (X | Y))  
 ((OY X (Y | Z))  
 (ON X Z)),
```

означая:

1. X находится в отношении ON со списком, голова которого равна X.

2. X находится в отношении ON со списком, состоящим из головы Y и хвоста Z, если X находится в отношении ON с Z.

Отношение APPEND определяется так:

```
((APPEND ( ) X X))  
((APPEND (X Y) Z (X x)) (APPEND Y Z x))
```

Это означает:

1. Результатом объединения пустого списка () с любым списком X является список X.

2. Если результатом объединения списков Y и Z является список x, то результатом объединения списков X | Y и Z будет список X | x, т. е. отношение APPEND используется как для объединения, так и для расчленения списков.

Например, в ответ на запрос

```
which (Z: APPEND (John Joe) (Alan Bill) x))
```

будет получено (John Joe Alan Bill), т. е. производится объединение двух списков.

Далее, ответом на запрос

which (x: APPEND (x (D E) (A B C D E)))

является (A B C),

т. е. списку x будет присвоено значение (A B C).

Наконец, запрос

which (x: APPEND ((a b c) x (a b c Joe)))

приведет к тому, что x получит значение (Joe).

Упражнение 3.1

Определить, что будет получено в ответ на следующие запросы:

a) all (x: x ON (1/24) () a2 2a)

b) all (x: x ON (ab abc (a (Joe Ted) John) def)

Упражнение 3.2

Определить результаты:

a) which (x: APPEND ((THIS IS) (A LIST) x))

b) which (x: APPEND (x (IS) (WHAT IS)))

в) which (x: APPEND ((THIS) x (THIS IS)))

г) which (x y: APPEND ((a b) (c d e f) (x | y)))

д) which (x y: APPEND ((x | y) (e f) (ab cd e f)))

3.2. ИСПОЛЬЗОВАНИЕ СПИСКОВ В БАЗАХ ДАННЫХ

Теперь можно проверить ряд методов, использование которых, возможно, позволит справиться с трудностями, перечисленными в начале этой главы. Предположим, что надо создать базу данных для хранения информации, приведенной в табл. 3.1. Один из возможных путей заключается в объединении всей информации о каждой личности в список и использовании для работы с этим списком специального отношения, названного, например, stat.

Таблица 3.1

Имя	Возраст, лет	Профессия	Годовой доход, долл.	Стаж	Номер дома	Проживание в доме, лет
Алан	15	0	500	0	2	7
Билл	42	Мясник	12 000	23	19	7
Колин	28	Учитель	9 850	7	3	2
Диана	30	Санитарка	6 500	5	7	5
Ева	12	0	0	0	4	12
Фрэнсис	32	Врач	11 500	5	2	2

Заметим, что в принципе можно использовать любое другое имя, но лучше выбрать такое имя, которое мнемонически отражает смысл отношения. В нашем случае stat — сокращение status (общественное положение) или statistics (статистика). Ниже приведено описание отношения на языке Пролог:

```

Alan stat (15 0 500 0 2 7)
Bill stat (32 butcher 12000 23 19 7)
Colin stat (28 teacher 9850 7 3 2)
Diana stat (30 nurse 6500 5 7 5)
Eve stat (12 0 0 0 4 12)
Frances stat (32 doctor 11500 5 2 2)

```

В отношении присутствуют 36 единиц информации (или в случае учета имени 42 единицы).

Шесть строк нашего отношения в принципе заменяют 36 строк, которые были бы необходимы, если бы для представления каждой единицы информации использовалось отдельное утверждение. Но, правда, нужно учесть, что несколько дополнительных утверждений потребуется для того, чтобы иметь возможность выделить из списка нужный элемент данных. Приведенные ниже отношения как раз используются для этой цели.

```

X aged Y if
    X stat (Y Z)
X job Y if
    X stat (Z Y x) and
    not X stat (Z 0 x)
X earns Y if
    X stat (Z x Y y)
X years-at Y if
    X stat (Z x y Y z)
X house Y if
    X stat (Z x y z Y X1)
X years-in Y if
    X stat (Z x y z X1 Y Y1)

```

Эта программа, без сомнения, более компактна и проще поддается изменениям, чем аналогичная программа, описанная в конце гл. 2. Достаточно легко, например, добавить к ней новые элементы данных. Для этого надо поместить их в конец списка и предусмотреть новое правило для их извлечения. Забегая немного вперед, скажем, что утверждения типа

```

Alan stat (15 0 5000 0 2 7)

```

в теории баз данных называются записями. Все элементы данных, входящие в список, носят названия полей; а имя человека, составляющего вместе со списком отношение, ключом записи.

В том случае, когда к какой-нибудь записи добавляются поля, необходимо модифицировать все имеющиеся записи данного типа, невзирая на то, что новые поля могут не входить в состав одной

или нескольких записей. Именно по этой причине значениями некоторых полей в программе являются нули. Пролог сопоставляет переменные с константами с учетом их позиций в списке. Образец должен быть тоже определен, если позиция переменной Y, представляющей требуемое поле, сравнивается с позицией искомой информации в отношении stat. Например, предложения

Bill stat (32 butcher 12000 23 19 7)

и

Н job Y if X stat (Z Y | x)

используются для сопоставления Y с butcher в ответ на запрос which (x: Bill job x)

В последнем правиле программы используется stat (Z x y z X1 Y | Y1), а не stat (Z x y z X1 | Y), несмотря на то, что последняя конструкция также позволяет извлечь нужную информацию. Дело в том, что первая конструкция будет работать даже при добавлении к записи новых полей, а вторую — в этом случае потребуется модифицировать. Важно обеспечить программе гибкость; недостаток предусмотрительности может привести либо к ошибкам при извлечении информации из базы данных, либо к необходимости проведения дополнительных работ по перепрограммированию уже написанных компонентов.

Упражнение 3.3

Составьте запросы, с помощью которых можно получить:

- а) всю информацию из базы данных;
- б) название профессии Билла;
- в) фамилии, профессии и годовой доход тех, кому больше 30 лет;
- г) всю информацию о Фрэнсисе;
- д) фамилии и число лет проживания в доме тех, кто одинаковое время жил в одном и том же доме.

Рекомендуем проверить правильность Ваших ответов на ЭВМ. Предупреждаем, что задание не такое простое, как кажется на первый взгляд.

Тонкость последнего задания упражнения 3.3 заключается в том, что гораздо легче получить больше информации, чем требуется. Первая попытка выполнить упражнение может быть такой:

all (X Y Z: X years-in Z and Y years-in Z)

Другими словами, ищутся X и Y такие, которые прожили в своих домах одинаковое число лет Z.

В качестве ответа на этот запрос получим

Alan	Alan	7
Alan	Bill	7
Bill	Bill	7

и т. д.

Повторение одного и того же имени объясняется тем, что Пролог-система «не знает», что X и Y не должны принимать одно и то же значение. Кроме того, одним из возможных ответов будет

Bill Alan 7

в принципе дублирующий ответ

Alan Bill 7

С точки зрения формальной логики вполне справедливы следующие утверждения: Алан живет в своем доме столько же лет, сколько Билл живет в своем; если Алан живет в своем доме то же самое время, что и Билл живет в своем, то Билл живет в своем доме столько же лет, сколько Алан живет в своем.

Из всего написанного выше ясно, что необходимо, во-первых, чтобы X и Y отличались друг от друга и, во-вторых, чтобы информация (X Y) считалась тождественной информации (Y X). Да, воистину логика иногда раздражающе логична. Запрос же теперь можно составить так:

all (X Y Z: X years-in Z and Y years-in Z and X LESS Y)

Ответами на этот запрос будут

Alan Bill 7

и

Colin Frances 2

Напомним, что с помощью отношения LESS можно сравнивать строки символов в соответствии с кодами символов в ASCII. Это свойство очень полезно для сортировки.

Упражнение 3.4

Дополните описанные выше записи о людях следующими данными:

- а) главным увлечением;
- б) типом автомобиля;
- в) названием любимого музыкального жанра;
- г) составьте новые отношения, которые позволяли бы извлекать только что указанную информацию.

3.3. СПИСКИ, СОСТОЯЩИЕ ИЗ СПИСКОВ

Те, кому удалось справиться с упражнением 3.4, наверняка заметил, что списковая структура с девятью элементами довольно громоздка. Рассмотрим типичный пример отношения stat:

Bill stat (42 butcher 12000 23 19 7 golf Metro jazz)

Сама по себе подобная форма представления информации не так уж плоха. Но для извлечения, например, данных об автомобиле требуется следующее предложение:

X car Y if

X stat (Z x y z X1 Y1 Z1 Y x1)

В результате значение, подлежащее определению, будет присвоено переменной Y. Но данное предложение будет работать, пока списки в базе данных содержат девять членов и никак не больше.

Указанный недостаток можно преодолеть, если составлять список из нескольких подсписков, в каждый из которых включать небольшое число членов. Если в какой-то момент времени под-списков становится слишком много, их, в свою очередь, можно объединить в подсписки более высокого уровня и т. д. С шестью подсписками, состоящими не более чем из шести элементов, работать довольно просто. А ведь это позволяет организовать доступ к 36 элементам данных. В интересах программиста включать в один и тот же подсписок данные, семантически связанные друг с другом. В нашем примере целесообразно один подсписок связать с профессией и включить в него специальность, годовой доход и стаж; другой подсписок будет содержать данные о склонностях человека: хобби, любимом музыкальном жанре и т. д. Это сделает программу легкой для понимания и удобной для использования и модификации. Более того, пользователям необязательно даже знать, как различные элементы объединяются в списки; они должны уметь только составлять запросы.

Предположим, что о каждом человеке надо хранить следующую информацию: возраст, дату и место рождения, адрес, профессию, годовой доход, стаж, название места работы и ее адрес, имя супруги (супруга), ее (его) возраст и число совместно прожитых лет. Будем считать всех интересующих нас людей совершеннолетними. Это не приведет к потере общности, поскольку в случае необходимости можно определить отношение, позволяющее отделить совершеннолетних от детей. Всю указанную информацию, представляющую собой смесь числовых и символьных данных, можно хранить в списках следующей структуры:

Bill stat ((42 Jan-25-1943 Leeds) (19 West-St Leeds) (butcher 1200 23)
(self 8 York-Rd Leeds) (Jane 39 18))

Tom stat ((24 Nov-3-1960 Dover) (7 Eglinton-St Andover) (engineer
8000 3) (Ace Engineers 17 Sprocket-St Salisbury) ())

Ted stat ((31 Mar-9-1954 Bath) (2 Brook-St Bath) () () (Laura 30 5))

Helen stat ((29 Aug-30-1956 Merton) (15 Albert-St Merton) (typist
6500 3) (Hill-and-Son 6 Nelson-Rd Wimbledon) (Jack 29 5))

Joe stat ((67 Oct-9-1917 Bristol) (23 South-Rd Cheltenham) (retired ()
5) () (Sarah 64 41))

Betty stat ((35 May-3-1950 Larne) (18 Antrim-Rd Larne) (nurse 9500 13)
(St-Marys-Hospital 226 Longpark Belfast) (James 38 13))

Читатели имеют возможность заменять любые записи и добавлять нужные им данные. Информация хранится в следующей форме:

(ключ) (отношение) ((список 1) (список 2) (список 3)
(список 4) (список 5))

Извлечь любой нужный подзаписок можно с помощью правил, аналогичных приведенным ниже:

X birth Y if
X stat (Y|Z)
X address Y if
X stat (Z Y|x)

В данном случае переменные будут сопоставляться не с отдельными элементами данных, а с целыми подзаписками. Так, согласно первому правилу переменная Y будет сопоставляться с подзаписком, содержащим дату и место рождения. Запрос, позволяющий получить эту информацию, может выглядеть так:

all (x y: x birth y)

В ответ на этот запрос система сообщает

Bill (42 Jan-25-1943 Leeds)

[Билл (42 25 января 1943 года Лидс)]

и т. д.

Чтобы извлечь нужный элемент из подзаписки, можно использовать правила типа

X age Y if
X stat ((Y | Z) | x)

Это правило позволяет получить информацию о возрасте, которая находится в первом элементе первого списка. Информацию о профессии можно извлечь с помощью следующего правила:

X job Y if
X stat (Z x (Y | y) | z)

Таким образом, данные о профессии находятся в первом элементе третьего списка. Читатели теперь без труда составят отно-

шения, предназначенные для извлечения всей остальной информации.

Упражнение 3.5

а. Составьте отношения для извлечения оставшихся подписков.

б. Определите, сколько элементов находится в каждом из пяти списков, и напишите отношения для доступа к этим элементам.

Упражнение 3.6

Составьте отношения для определения:

а. Кто состоял в браке одно и то же число лет?

б. У кого одинаковый стаж?

в. У кого стаж совпадает с числом лет, прожитых в браке?

г. Кто живет там, где родился?

д. Кто живет и работает там, где родился?

Тем читателям, которые хотят самостоятельно выполнить упражнения 3.5 и 3.6, рекомендуется пока не заглядывать дальше.

Поскольку каждый обобщенный список состоит из пяти подписков, для доступа к ним необходимо пять правил. Ниже приведены эти правила, хотя в принципе читатели могут выбрать для отношений любые другие имена:

```
X birth Y if
    X stat (Y|Z)
X home Y if
    X stat (Z Y|x)
X work Y if
    X stat (Z x Y|y)
X firm Y if
    X stat (Z x y Y|y)
X marriage Y if
    X stat (Z x y z Y|X1)
```

С помощью каждого из этих правил можно получить сразу всю информацию, относящуюся к какой-то одной стороне личности. Например, запрос

all (x y: x firm y)

позволяет определить имена всех тех, кто фигурирует в базе данных вместе с информацией о месте их работы.

Подписки содержат 3, 3, 3, 4 и 3 элемента соответственно. Таким образом, общее число элементов данных равно 16. Очевидно, что для извлечения этих элементов потребуется 16 правил. Отметим, что в четвертом подписке содержится четыре элемента:

название фирмы, номер дома, название улицы, город. Вот эти 16 правил:

$X \text{ aged } Y \text{ if}$
 $\quad X \text{ birth } (Y \mid Z)$
 $X \text{ birthdate } Y \text{ if}$
 $\quad X \text{ birth } (Z \mid Y \mid x)$
 $X \text{ birthplace } Y \text{ if}$
 $\quad X \text{ birth } (Z \times Y \mid y)$
 $X \text{ home-no } Y \text{ if}$
 $\quad X \text{ home } (Y \mid Z)$
 $X \text{ home-st } Y \text{ if}$
 $\quad X \text{ home } (Z \mid Y \mid x)$
 $X \text{ home-place } Y \text{ if}$
 $\quad X \text{ home } (Z \times Y \mid y)$
 $X \text{ job } Y \text{ if}$
 $\quad X \text{ work } (Y \mid Z)$
 $X \text{ earns } Y \text{ if}$
 $\quad X \text{ work } (Z \mid Y \mid x)$
 $X \text{ years-in-job } Y \text{ if}$
 $\quad X \text{ work } (Z \times Y \mid y)$
 $X \text{ employer } Y \text{ if}$
 $\quad X \text{ firm } (Y \mid Z)$
 $X \text{ work-no } Y \text{ if}$
 $\quad X \text{ firm } (Z \mid Y \mid x)$
 $X \text{ work-st } Y \text{ if}$
 $\quad X \text{ firm } (Z \times Y \mid y)$
 $X \text{ workplace } Y \text{ if}$
 $\quad X \text{ firm } (Z \times Y \mid z)$
 $X \text{ spouse } Y \text{ if}$
 $\quad X \text{ marriage } (Y \mid Z)$
 $X \text{ spouse-age } Y \text{ if}$
 $\quad X \text{ marriage } (Z \mid Y \mid x)$
 $X \text{ married } Y \text{ if}$
 $\quad X \text{ marriage } (Z \times Y \mid y)$

В них использованы ранее определенные отношения, предназначенные для извлечения сразу всех данных, принадлежащих подписку. Отметим, что можно получить данные из большего по размеру списка с помощью отношений вида

$X \text{ aged } Y \text{ if}$
 $\quad X \text{ stat } ((Y \mid Z) \mid x)$

Это последнее правило, ссылающееся на отношение *stat*, может быть быстрее выполнено, поскольку в нем фигурируют только два отношения. С другой стороны, метод, заключающийся в использовании дополнительных правил для извлечения элементов данных из подписки, определенного с помощью отношения *stat*, оперирует с более коротким списком неизвестных. Особенно это заметно для подписков, расположенных в конце главного списка. Кроме того, преимущество этого метода также в том, что образ неизвестного списка одинаков сразу для нескольких правил.

Чтобы убедиться в этом, достаточно сравнить правила, описывающие возраст и дату рождения, с правилами, описывающими место жительства.

Таким образом, программа, реализующая этот метод, во-первых, удобнее для понимания и, во-вторых, в случае добавления новых данных соответствующие им правила будут иметь ту же самую структуру, что и уже имеющиеся в программе правила. Для включения новых правил можно использовать команду `cedit`, упомянутую во второй главе. Отметим, что во всех выше описанных отношениях предусмотрена возможность расширения базы данных. Именно поэтому отношение `marriage` определяется так, как показано ниже:

```
X marriage Y if
  X stat (Z x y z Y | X1)
```

а не так:

```
X marriage Y if
  X stat (Z x y z Y)
```

Последнее правило дает возможность извлекать нужные данные, так как `Y` является последним подписанием в главном списке. Но при добавлении новых подписок это правило надо модифицировать. Аналогично правило

```
X homeplace Y if
  X home (Z x Y | y)
```

хотя и позволяет определить город, в котором живет тот или иной человек, но проигрывает по сравнению с правилом

```
X homeplace Y if
  X home (Z x Y | y)
```

поскольку последнее не требует модификации в случае пополнения данных, принадлежащих подписке.

Ясно, что много полезной информации может быть получено в результате анализа данных, находящихся в подписках. Ниже дано несколько примеров, позволяющих получить эту информацию. Предположим, что необходимо найти всех тех, кто работает в том же городе, в котором живет:

```
all (x y: x homeplace y and x workplace y)
[определить все (x y: x живет в городе y и x работает в городе y)]
Bill Leeds
[Билл Лидс]
No (more) answers
[Ответов (больше) нет].
```

Однако попытка определить всех тех, кто работает не там, где живет, с помощью запроса

```
all (x y z: x homeplace y and x workplace z)
```

приводит к неудаче. В ответ на этот запрос будут получены следующие данные:

Bill Leeds Leeds
[Билл Лидс Лидс]
Tom Andover Dover
[Том Эндувр Дувр]
Helen Merton Wimbledon
[Эллен Мертон Унмблдон]
Betty Larne Belfast
[Бетти Лан Белфаст]
No (more) answers
[Ответов (больше) нет]

Понятно, что это не совсем то, что хотелось. Попробуем проанализировать последний запрос. Кажется, что поскольку две переменные u и z используются для названий городов, программа должна сопоставить им разные значения. Но это не так. Сначала программа ищет, какие константы можно сопоставить переменным x и y , причем эти константы должны входить в отношение `homeplace`. В результате x будет поставлено в соответствие `Bill`, а y — `Leeds`. Затем программа ищет пример отношения `workplace`, первым аргументом которого является `Bill`. После успешного завершения поиска несвязанной переменной присваивается значение `Leeds`. И, наконец, выводится первый из приведенных выше ответов. Заметим, что в запросе нет никакой информации о том, что u и z не могут принимать одно и то же значение. В процессе обработки запроса для хранения значений переменных x , y и z выделяется специальная структура, называемая стеком. После того, как первое решение найдено, значение z выталкивается из стека, и программа старается сопоставить с z другое значение. Процесс поиска альтернативных решений называется поиском с возвратом или бэктрекингом. Для переменной z на данном шаге других альтернативных значений не существует. Поэтому программа возвращается к оценке первой части запроса `Bill homeplace y`. Поскольку новых значений для переменной y не существует, вся описанная процедура повторяется, только в качестве x теперь используется значение `Tom`.

Стек представляет собой структуру данных, которая занимает ряд смежных позиций памяти. Очередная порция данных помещается в стек с помощью операции `PUSH`, а удаляется из стека с помощью операции `POP`. Стек часто называют структурой типа «последним пришел — первым ушел»*. Это означает, что порция данных, помещенная в стек последней, удаляется первой. Функционирование стека напоминает работу автоматической раздаточной машины; отличие состоит в том, что данные и добавляются

* Часто используется аббревиатура LIFO (last in first out) — Прим. пер.

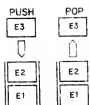


Рис. 3.1. Операции со стеком:
PUSH — помещение элемента в стек; POP — выталкивание элемента из стека

в стек и извлекаются из стека пользователями. Переменная вершина стека указывает на позицию, в которой размещается последний помещенный в стек элемент данных. Во многих ЭВМ стек реализован так, что «растет» сверху вниз, т. е. положение вершины стека остается неизменным. Но все это в принципе не касается тех, кто программирует на Прологе.

На рис. 3.1 изображены диаграммы, иллюстрирующие выполнение операций со стеком; элемент E3 помещается в стек последним — следом за элементами E1 и E2, а выталкивается первым. Операции со стеком широко используются Пролог-системой в процессе оценки применимости правил. В связи с этим программист должен сознавать, что существует опасность чрезмерного роста стека, и принимать необходимые меры для ее устранения. К более детальному рассмотрению этого вопроса мы перейдем позднее.

Упражнение 3.7

Вернемся к обсуждавшемуся ранее вопросу. Может быть запрос

all (x y: x homeplace y and not x workplace y)

позволит решить проблему? Если для проверки Вы используете ЭВМ, то поймете, что это не так. Но тем не менее этот запрос очень похож на тот, который нам нужен.

Упражнение 3.8

Составьте запросы, позволяющие определить.

- а. Кто зарабатывает более 8000 долл. и каков их точный годовой доход?
- б. Каковы имена людей старше 30 лет, чей годовой доход менее 10 000 долл.?
- в. Кто работает у себя дома?
- г. Кто дольше, чем Том, работает на одном месте?
- д. Кто женат дольше, чем Джо?

3.4. ПРАВИЛА ИЛИ ЗНАНИЯ

В последней программе хранятся данные как о возрасте, так и дате рождения каждого человека. Но достаточно в принципе задать только дату рождения, а в программе предусмотреть правила, позволяющие по вводимой дате вычислять возраст. Этот пример достаточно характерен и позволяет сформулировать важ-

ную задачу, всякий раз возникающую перед программистом: какую программную систему разрабатывать — основанную на правилах или на знаниях? Отметим, что в чистом виде систем, основанных только на знаниях или только на правилах, не существует. Но могут существовать как программы, включающие очень немного правил и много фактов, так и программы, состоящие в основном из правил. Программы, основанные на знаниях, обычно работают быстрее, чем аналогичные программы, базирующиеся на правилах, но зато для хранения дополнительных данных требуется лишняя память.

В качестве примера рассмотрим новый формат отношения stat:

Bill stat ((25 1 1943 Leeds) ...)

Многообразие свидетельствует о том, что оставшаяся часть отношения не изменилась. К каждому из элементов данных в принципе легко обратиться, но для определения возраста теперь требуется дополнительное отношение. Кроме того, должна быть известна текущая дата.

Новые правила, предназначенные для работы с данными о дате рождения, будут теперь иметь вид

```

X birth Y if
  X stat (Y | Z)
X birthdate (Y Z x) if
  X birth (Y Z x | y)
X birthplace Y if
  X birth (Z x y Y | z)

```

Обратите внимание на то, что в правилах предусмотрена возможность увеличения числа данных.

Отметим, что задача определения возраста сложнее, чем может показаться на первый взгляд. Необходимо учесть следующие соображения:

если номер текущего месяца больше номера месяца даты рождения, то необходимо просто вычесть год рождения из текущего года;

ту же самую операцию надо проделать, если номера месяцев совпадают, а номер текущего дня больше или равен номеру дня рождения;

если же номер месяца даты рождения больше номера текущего месяца или номера месяцев совпадают, а номер дня рождения больше номера текущего дня, то необходимо вычесть год рождения из текущего года и результат уменьшить на единицу.

Напомним, что отношение меньше или равно имеет следующий вид:

$X \leq X$

$X \leq Y$ if

$X \text{ LESS } Y$

Ниже приведены правила, позволяющие вычислить возраст человека:

X age Y if

**X birthdate (Z x y) and
date now (z X1 Y1) and
(either x LESS X1 and / or x leq X1 and Z leq z) and
SUM (Y y Y1)**

X age Y if

**X birthdate (Z x y) and
date now (z X1 Y1) and
(either X1 LESS x and / or X1 leq x and z LESS Z) and
SUM (Z1 y Y1) and
SUM (Y 1 Z1)**

Символ «/» используется здесь для обозначения специальной команды, которая запрещает производить возврат для поиска альтернативных решений. Эту команду нам предстоит рассмотреть более подробно, но сделаем мы это немного позднее. Сейчас же будем считать, что любое условие, стоящее перед «/», будет анализироваться только один раз.

Отношение SUM дважды используется для определения разности между текущим годом и годом рождения и один раз для вычитания единицы. Для проверки работы программы необходимо включить в ее состав отношение date и сформировать запрос, позволяющий определить возраст всех тех людей, информация о которых содержится в базе данных.

Для более тщательного тестирования программы рекомендуем в качестве текущей использовать дату, месяц и число, которое или совпадает с месяцем и числом чьей-нибудь даты рождения, или предшествует ей, или следует за ней. Хранить в базе данных следует только один экземпляр даты, в противном случае в ответ на каждый запрос будет получено несколько альтернативных ответов. Ниже приведен характерный пример диалога пользователя с системой:

date now (8 10 1985)

[текущая дата (8 10 1985)]

all (x y: x age y)

[определить все (x y: x имеет возраст y)]

Bill 42

Ted 31

Helen 29

Betty 35

Tom 24

Joe 67

No (more) answers

[Ответов (больше) нет]

date now (9 10 1985)

[текущая дата (9 10 1985)]

all (x y: x age y)

[определить все (x y: x имеет возраст y)]

Bill 42

Ted 31

Helen 29

Joe 68

Betty 35

Tom 24

No (more) answers

[Ответов (больше) нет]

date now (21 3 1985)

[текущая дата (21 2 1985)]

all (x born on y and now aged z: x birthdate y and x age z)

[определить все (Дата рождения x-у и его возраст в данный момент z: x родился у и x имеет возраст z)]

Bill born on (25 1 1943) and now aged 42

[Дата рождения Билла — (25 1 1943) и его возраст в данный момент 42]

Tom born on (3 11 1960) and now aged 24

[Дата рождения Тома — (3 11 1960) и его возраст в данный момент — 24]

Ted born on (9 3 1954) and now aged 31

[Дата рождения Теда — (9 3 1954) и его возраст в данный момент — 24]

Helen born on (30 8 1956) and now aged 28

[Дата рождения Элен — (30 8 1956) и ее возраст в данный момент — 28]

Joe born on (9 10 1917) and now aged 67

[Дата рождения Джо — (9 10 1917) и его возраст в данный момент — 67]

Betty born on (3 5 1950) and now aged 34

[Дата рождения Бетти — (3 5 1950) и ее возраст в данный момент — 34]

No (more) answers

[Ответов (больше) нет]

3.5. МОДИФИКАЦИЯ ПРОГРАММЫ

В рассматриваемую нами программу должны время от времени вноситься изменения. Укажем, в каких случаях это необходимо.

1. В базу данных требуется добавлять информацию о новом человеке или удалить всю информацию о той или иной личности. Эти операции достаточно просто выполнить с помощью отношений `add` и `delete`.

2. В базу данных может потребоваться ввести дополнительную информацию о личности; причем ее следует поместить либо в один существующий подписок, либо в несколько существующих подписков, либо потребуется создать новый подписок и включить данные в него. Такую возможность необходимо принять во внимание при конструировании правил, предназначенных для доступа к уже имеющейся в базе данных информации. Все изменения должны быть сведены только к включению в программу дополнительной информации и новых правил для доступа к ней.

3. Довольно часто требуется изменять данные, хранящиеся в базе. Характерным примером таких данных является текущая дата.

Отметим, что устройство, отсчитывающее время, или таймер, обычно полностью автономно и не прекращает своей работы даже тогда, когда отключается питание ЭВМ. Чтобы определить дату, достаточно обратиться к этому устройству. Именно это сразу после включения ЭВМ делает системная программа инициализации. Кроме того, существуют следующие способы определения даты: с помощью специальной программы, которая вызывается в тех случаях, когда для обработки запроса требуется дата; с помощью отношения, которое либо запрашивает у пользователя дату, либо побуждает ЭВМ модифицировать дату в соответствии с некоторым фиксированным правилом. Поскольку последним способом наиболее часто пользуются владельцы микроЭВМ, мы рассмотрим несколько возможных путей его реализации.

Читателям уже известен используемый в Пролог-системе способ модификации информации, основанный на отношении `is-told`. Для того чтобы воспользоваться им, необходимо загрузить в оперативную память модуль `TOLD`.

Напомним, что иногда текст программы занимает слишком много места и в связи с этим работа программы замедляется. Чтобы этого не было, следует после отладки программы удалить модули системы `SIMPLE` из памяти и все запросы формулировать, следуя требованиям стандартного синтаксиса. Однако удобно постоянно иметь в распоряжении систему `SIMPLE`. Поэтому, для того чтобы все же ускорить работу, мы введем в рассмотрение новую программу, которая связана главным образом с использованием даты для получения ответов на запросы.

Читателю, который совмещает чтение этой книги с работой на ЭВМ, к сожалению, придется удалить прежнюю программу и набрать новую, текст которой приведен ниже:

```

Betty born (3 5 1950)
Joe born (9 10 1917)
Helen born (30 3 1950)
Tom born (3 11 1960)
Bill born (25 1 1960)
Ted born (9 3 1954)
Mary born (1 1 1930)
Isabel born (31 12 1942)
X age Y if
    X born (Z x y) and
    date now (z X1 Y1) and
    (either x LESS X1 and / or x leq X1 and Z leq z) and
    SUM (Y y Y1)
X age Y if
    X born (Z x y) and
    date now (z X1 Y1) and
    (either X1 LESS x and / or X1 leq x and z LESS Z) and
    SUM (Z1 y Y1) and
    SUM (Y 1 Z1)
X leq X
X leq Y if
    X LESS Y
date now (20 9 1985)

```

В ответ на запрос

all (x y z: x born y and x age z)

должны быть получены следующие ответы:

```

Betty ( 3 5 1950) 35
Joe ( 9 10 1917) 67
Helen (30 8 1956) 29
Tom ( 3 11 1950) 34
Bill (25 1 1943) 42
Ted ( 9 3 1954) 31
Mary ( 1 1 1980) 5
Isabel (31 12 1942) 42

```

Теперь сформулируем правило, позволяющее модифицировать дату:

```

update X if
    now KILL and
    (new date (X)) is-told and
    (date now X) add and

```

Продемонстрируем на примере, как оно работает:

```
is (update x)
[верно (модифицировать x)]
new date (X) ? ans (19 9 1985)
[новая дата (X) ? ответ (19 9 1985)]
Yes
[Да]
```

Когда текст программы будет выведен на печать, окажется, что старая дата уничтожена, а новая добавлена. Следует вводить только одну новую дату, поскольку все равно отношение «/» предотвратит использование всех дат, кроме одной. Если эта предосторожность не будет выполнена, новые даты будут запрашиваться до тех пор, пока от пользователя не будет получен ответ NO. Заметим, что при вводе необходимо дату заключать в круглые скобки.

Не прибегая к модификации отношения age, сформируем новое отношение, которое позволит одновременно изменить дату и определить возраст. Когда дату корректировать не надо, можно использовать описанные выше отношения:

```
X new-age Y if
  update Z and
  X age Y
```

Теперь программа будет работать следующим образом:

```
all (x y: x new-age y)
[определить все (x y: x имеет-новый-возраст y)]
new date (X) ? ans (12 12 1985)
[новая дата (X) ? ответ (12 12 1985)]
Betty 35
[Бетти 35]
Joe 68
[Джо 68]
и т. д.
```

Упражнение 3.9

Составьте запросы, позволяющие определить:

- Кто моложе, чем Тед, и каков его возраст?
- Кому 1 января 1975 г. было больше 18 лет?
- Кому 31 декабря 2000 г. будет меньше 50 лет?
- Кто в данный момент имеет один и тот же возраст?
- У кого даты рождения совпадают?
- Кто родился позднее 1 января 1950 г.?

Упражнение 3.10

В табл. 3.2 приведены запасы стройматериалов, которыми располагает три строительных подрядчика.

Т а б л и ц а 3.2

Подрядчик	Песок, м	Цемент, т	Блоки, шт.	Кирпич, тыс. шт.
McDoo	30	12	200	555
Fec	598	100	3500	9760
Putlog	8900	950	7380	6875

Напишите программу, которая позволит пользователю определить, какое количество указанного материала находится у данного поставщика, и, кроме того, даст возможность переопределить ресурсы поставщиков после покупки и продажи материалов.

Программисту, привыкшему работать на Бейсике, программа на Прологе для упражнения 3.10 покажется громоздкой и какой-то нескладной. На Бейсике все выглядело бы значительно проще: достаточно было бы нескольких операторов типа LET или одного FOR. Но в Прологе таких возможностей нет. Существует принципиальная разница в методах обработки данных, используемых Бейсиком и Прологом. В Бейсике переменные принимают глобальные значения, которые в процессе выполнения программы меняются в соответствии с выполняемыми операциями; в Прологе переменные не принимают глобальных значений и с ними связываются значения только в результате обработки запроса. Кроме того, в Прологе после обработки запроса последнее назначенное переменной значение не запоминается.

В языках, подобных Бейсику, данные являются динамическими или изменяемыми; в то же время данные в Прологе статичны или неизменны. Одна из ловушек, подстерегающих в Прологе тех, кто привык к Бейсику, заключается в том, что значения, вводимые с помощью отношения is-told, не сохраняются. Для того чтобы сохранить значения, необходимо использовать отношение add; для удаления старых значений можно применять отношение delete. Эти два отношения можно использовать для того, чтобы определить, сколько раз применялось заданное правило в процессе обработки запроса. Отметим, что эту же задачу на Бейсике решить несколько проще, чем на Прологе.

Предположим, требуется узнать, сколько раз программа использует дату в процессе обработки запроса о возрасте. Для этого необходимо создать два отношения: одно — для хранения требуемого числа и второе — для его модификации. Кроме того, для того чтобы предусмотреть модификацию информации, необходимо добавить условие к правилу, использование которого контролируется.

В результате изменения, вносимые в программу, будут иметь вид:

```
date now (22 9 1985) if
  A upcount
X upcount if
  (X count Y) delete and
  SUM (Y 1 Z) and
  (X count Z) add and
  /
  A count 0
  B count 0
  C count 0
  D count 0
```

Выполнение условия

if A upcount

приводит к замене предложения

A count N

предложением

A count N + 1

всякий раз, когда используется отношение date.

Остальные утверждения, включающие объекты B, C и D, потребуются, если понадобится контролировать какие-то другие отношения.

На запрос

all (x A count y : x age z and A count y)

теперь будут получены следующие ответы:

Betty	A count	1
Helen	A count	3
Bill	A count	5
Ted	A count	6
Mary	A count	7
Joy	A count	10
Tom	A count	12

Подчеркнем, что отношение count позволяет определить общее число обращений к дате. Для Бетти, Теда и Мэри число таких обращений равно 1, для Элен, Билла и Тома — 2, для Джо — 3 и для Изабеллы — 4. Можно предположить, что программа, включающая отношения count и update, будет работать медленно.

Это действительно так, поскольку выполнение операций, связанных с отношениями `add` и `delete`, потребует значительного времени. Другими словами, описанный метод стоит использовать только для выполнения диагностических функций. Более того, в Пролог-системе есть специальный модуль `TRACE`, позволяющий следить за каждым шагом работы программы в процессе оценки запроса. Для того чтобы осуществить это, необходимо разместить модуль `TRACE` непосредственно перед программой пользователя. Но следует учесть, что он занимает достаточно много памяти и поэтому его целесообразно использовать как средство обучения только вместе с небольшими программами. Если с помощью отношения `count` необходимо следить за работой нескольких правил, то было бы удобно иметь универсальное средство установки в ноль значений всех счетчиков, использование которого даст возможность избежать модификации каждого предложения. Ниже приведена программа, которая позволяет это сделать. В ней предусмотрено удаление всех имеющихся предложений отношения `count` и добавление новых для счетчиков `A`, `B` и `C`

```

X zero if
  (X count Y) delete and
  (X count 0) add and
  /
X reset if
  Y isall (Y: Y count Z) and
  X ON Y and
  not X count 0 and
  X zero
A count 5
B count 55
C count 23

```

Для установки в ноль всех счетчиков достаточно использовать следующий запрос

```
all (x: x reset)
```

В ответ будет напечатано `C`, `B` и `A` — именно в таком порядке. Если Вы теперь распечатаете текст программы, то увидите, что все счетчики установлены в ноль. Заметим, что отношение `zero` позволяет удалить из базы данных только одно утверждение вида

```
X count Y
```

Нам же необходимо удалить все. Для этого предлагается использовать стандартное отношение `isall`. Это отношение в данном примере дает возможность сформировать список `Y`, состоящий из всех первых аргументов отношения `count`. Причем порядок следования этих аргументов изменится на противоположный, т. е.

$Y = (C \vee A)$. Для ускорения работы программы выполнение отношения `reset` иницируется только тогда, когда данный счетчик не установлен в ноль, т. е. когда второй аргумент утверждения `count` не равен нулю.

Упражнение 3.11

а. Удалите условие

`If A ureount`

из отношения `date` и добавьте его к первому правилу отношения `age`. Кроме того, добавьте второй счетчик `B` ко второму правилу отношения `age`.

б. Составьте запрос, который даст возможность определить, сколько раз каждое правило, образующее отношение `age`, используется для определения возраста всех тех людей, информация о которых имеется в базе данных.

в. Попробуйте объяснить, почему данные о возрасте людей не выводятся в том же порядке, в котором расположены имена этих людей в базе данных.

3.6. УВЕЛИЧЕНИЕ СКОРОСТИ И ЭФФЕКТИВНОСТИ ПРОГРАММ

Изложим ряд соображений, которые могут быть сделаны на основе анализа использования счетчиков в предыдущей программе.

1. При определении возраста частота применения тех или иных правил зависит от того, чей возраст определяется.

2. Последовательность размещения выводимых на печать результатов не совпадает с последовательностью размещения информации в базе данных.

3. Последовательность размещения результатов может измениться, если изменится текущая дата. Проверить это достаточно просто; для этого необходимо следующим образом выбирать текущую дату: сначала она должна предшествовать дате какого-нибудь дня рождения, затем совпадать с ней и, наконец, следовать за ней. Но учтите, что в каждый момент времени в базе данных должна находиться только одна дата.

4. Последовательность размещения результатов изменится, если поменять местами первое и второе утверждения отношения `age`.

Значительно большими диагностическими возможностями обладают специальные программы трассировки, входящие в состав системы программирования микроПролог. Максимально возможную информацию обеспечивает, например, программа `SIMTRACE`. Поскольку для этой программы требуется довольно много па-

мяти, перед ее использованием удалите из памяти все, что там было, и введите программу, приведенную ниже.

```
Betty born (3 5 1950)
Joe born (23 9 1917)
Bill born (22 9 1943)
Isabel born (21 9 1942)
X age Y if
    X born (Z x y) and
    date now (z X1 Y1) and
    (either x LESS X1 and / or x leq X1 and Z leq z) and
    SUM (Y y Y1)
X age Y if
    X born (Z x y) and
    date now (z X1 Y1) and
    (either X1 LESS x and / or X1 leq x and z LESS Z) and
    SUM (Z1 y Y1) and
    SUM (Y 1 Z1)
X leq X
X leq Y if
    X LESS Y
date now (22 9 1985)
```

Тестовый запрос для этой программы и ответы на него будут выглядеть так:

```
all (x y: x age y)
Betty 35
Bill 42
Isabel 45
Joe 67
No (more) answers
```

Даты рождения людей выбраны таким образом, чтобы они не посредственно предшествовали и следовали за текущей датой — (22 9 1985). Это будет гарантировать участие в обработке запроса всех правил. После того как у Вас появилась уверенность в правильности программы, удалите из памяти модули program-mod и errmsg-mod с помощью команды KILL. Затем используйте команду LOAD для того, чтобы загрузить на свободное место программу трассировки SIMTRACE. В том случае, если памяти все же окажется недостаточно, попробуйте удалить из модуля query-mod любые из следующих отношений: which, all, one, is, load, save, +, —, %, @, *, CONS, reserved, true-of и defined.

Естественно, после этого использовать в запросах те отношения, которые Вы удалили, будет нельзя. Для того чтобы получить полную информацию о процессе обработки запроса, введите

```
all-trace (x y: x age y)
```

Ниже приведен протокол диалога между пользователем и ЭВМ:

&all-trace (x y: x age y)
[& полная трассировка (x y: x имеет возраст y)
(1): X age Y trace ? y
[(1): X имеет возраст Y трассировка ? да]
matching (1) : X age Y with head of 1 : Z age x
[сопоставление (1) : X имеет возраст Y с головой правила 1 : Z имеет возраст x]
match succeeds : X age Y
[сопоставление завершено успешно : X имеет возраст Y]
new query : X born (Y Z x) and date now (y z X1) and
[новый запрос : X родился (Y Z x) и текущая дата (y z X1) и]
(either Z LESS z and/or Z leq z and Y leq y) and SUM (Y1 x X1)
[(либо Z LESS z and/либо Z меньше или равно z и Y меньше или равно y) и SUM (Y1 x X1)]
(1 1) X born (Y Z x) trace ? y
[(1 1) X родился (Y Z x) трассировка ? да]
matching (1 1) : X born (Y Z x) with head of 1 : Betty born
(3 5 1950)
[сопоставление (1 1) : X родился (Y Z x) с головой правила 1 : Бетти родилась (3 5 1950)]
match succeeds : Betty born (3 5 1950)
[сопоставление успешно завершено : Бетти родилась (3 5 1950)]
(1 1) solved : Betty born (3 5 1950)
[(1 1) результат сопоставления: Бетти родилась (3 5 1950)]
(2 1) : date now (X Y Z) trace ? y
[(2 1) : текущая дата (X Y Z) трассировка? да]
matching (2 1) : data now (X Y Z) with head of 1 : date now
(22 9 1985)
[сопоставление (2 1) : текущая дата (X Y Z) с головой правила 1 : текущая дата (22 9 1985)]
match succeeds : date now (22 9 1985)
[сопоставление успешно завершено : текущая дата (22 9 1985)]
(2 1) solved : date now (22 9 1985)
[(2 1) результат сопоставления : текущая дата (22 9 1985)]
(3 1) : (either 5 LESS 9 and/or 5 leq 9 and 3 leq 22) trace ? y
[(3 1) : (либо 5 LESS 9 и/либо 5 меньше или равно 9 и 3 меньше или равно 22) трассировка ? да]
(3 1) either branch
[(3 1) ветвление либо-либо]
(1 3 1) : 5 LESS 9
[(1 3 1) : 5 LESS 9]
(1 3 1) solved : 5 LESS 9
[(1 3 1) результат сопоставления : 5 LESS 9]
(2 3 1) : /
[(2 3 1) : /]
(2 3 1) solved : /


```

[(2 3 1) результат сопоставления : /]
(3 1) solved : (either 5 LESS 9 and/or 5 leq 9 and 3 leq 22)
[(3 1) результат сопоставления : (либо 5 LESS 9 и/либо 5
меньше или равно 9 и 3 меньше или равно 22)]
(4 1) : SUM (X 1950 1985)
[(4 1) : SUM (X 1950 1985)]
(4 1) solved : SUM (35 1950 1985)
[(4 1) результат сопоставления : SUM (35 1950 1985)]
(1 1) solved : Betty age 35
[(1 1) результат сопоставления : возраст Бетти — 35]
Betty 35
[Бетти 35]
backtracking ...
[возврат для поиска альтернативных вариантов ...]
(4 1) failing : SUM (X 1950 1985)
[(4 1) неудача : SUM (X 1950 1985)]
(3 1) failing : (either 5 LESS 9 and/or 5 leq 9 and 3 leq 22)
[(3 1) неудача : (либо 5 LESS 9 и/либо 5 меньше или равно 9
и 3 меньше или равно 22)]
retrying (2 1)
[снова попробовать (2 1)]
(2 1) failing: date now (X Y Z)
[(2 1) неудача : текущая дата (X Y Z)]
retrying (1 1)
[снова попробовать (1 1)]
matching (1 1) : X born (Y Z x) with head of 2 : Joe born
(23 9 1917)
[сопоставление (1 1) : X родился (Y Z x) с головой правила
2 : Джо родился (23 9 1917)]
match succeeds : Joe born (23 9 1917)
[сопоставление успешно завершено: Джо родился (23 9 1917)]
(1 1) solved: Joe born (23 9 1917)
[(1 1) результат сопоставления : Джо родился (23 9 1917)]
(2 1) date now (X Y Z) trace?
[(2 1) : текущая дата (X Y Z) трассировка?]

```

Многоточие в протоколе заменяет текст длинного условия, которое автору просто не захотелось здесь приводить. Наверное, читателю уже ясно, что пользователь должен вводить «у» в том случае, когда хочет продолжить трассировку. Как видно из протокола, сначала проверяется, существует ли в программе определение отношения, заданного в запросе; затем проверяется, имеется ли в программе отношение, являющееся первым в условии. Если одна из этих проверок оканчивается неудачей, выдается сообщение об ошибке. Но в данном случае все в порядке, и процесс трассировки продолжается.

Используемые во время трассировки утверждения и условия помечаются номерами, заключенными в скобки. Например, (1 1) —

первое утверждение в первом условии, (2 1) — второе утверждение в первом условии и т. д. Для предложений сложной структуры потребуется более двух номеров. Например, (1 3 1) — первая часть третьего утверждения в первом условии; в данном случае такой номер появляется, когда анализируется конструкция (either ... or...).

В результате проверки каждого условия формируется сообщение либо об успехе, либо о неудаче. В случае успеха решения помещаются в стек. Если весь процесс обработки запроса завершается успешно, программа выдает ответ и, используя механизм возврата, приступает к поиску альтернативных решений. В случае неудачи при проверке какого-то условия весь процесс только тогда считается неудачным, когда нет альтернативного варианта, задаваемого конструкцией (either ... or). Управление возвратом с помощью правильно сконструированных определений отношений и запросов позволяет программисту в ряде случаев значительно увеличить эффективность работы программы.

В нашем примере возврат для поиска альтернативных вариантов осуществляется в следующих случаях.

(4 1). Здесь поиск альтернатив для SUM (X 1950 1985) сразу же заканчивается неудачей, поскольку существует единственное значение X, для которого справедливо

$$X + 1950 = 1985.$$

(2 1). Здесь производится попытка найти отличный от (22 9 1985) список (X Y Z), который содержится в предложении вида

date now (X Y Z)

Эта попытка также заканчивается неудачей, так как такой список только один — (22 9 1985). Но если было бы введено несколько предложений, определяющих отношение date now, то все они использовались бы для поиска решений, связанных с определением возраста Бетти.

(3 1). Здесь ищется альтернативное решение для первой части конструкции (either ... or...).

Несколько слов необходимо сказать о том, как использование отношения «/» позволяет увеличить эффективность обработки запроса. Переход к отношению «/» означает, что возврат ко всем предшествующим условиям, сопоставление которых прошло успешно, запрещен. Кроме того, что также очень важно, ни одно из условий, стоящих после «/», в тех случаях, когда условия, предшествующие «/», доказаны, не анализируется. Пока в качестве главной причины неэффективности нашей программы можно отметить следующее: пытаюсь ответить на запрос X age Y, программа каждый раз осуществляет сопоставление

date now (X Y Z)

с соответствующим предложением программы и постоянно используется возврат для поиска альтернативных дат, которых заведомо нет. Это в принципе не слишком опасно, поскольку приводит к неудаче только один раз. Гораздо серьезнее следует относиться к возврату, который возникает в тех случаях, когда часть проверяемых условий удовлетворяется *. Отметим желательность выполнения какой-то операции, позволяющей программе оптимально организовывать доступ к текущей дате.

Если проследить за работой программы в процессе определения возврата Джо с помощью первого правила отношения аге, то можно увидеть причины, порождающие значительно большую неэффективность. Все в принципе хорошо до тех пор, пока не достигается условие (either ... or...). Процесс обработки запроса после достижения этого условия показан ниже:

```
(3 1) : either branch
[(3 1) ветвление либо-либо]
(1 3 1) : 9 LESS 9
[(1 3 1) : 9 LESS 9]
(1 3 1) : failing : 9 LESS 9
[(1 3 1) : неудача : 9 LESS 9]
(3 1) : or branch
[(3 1) : ветвление или]
(1 3 1) : 9 leq 9 trace ? y
[(1 3 1) : 9 меньше или равно 9 трассировка ? да]
matching (1 3 1) : 9 leq 9 with head of 1 : X leq X
[сопоставление (1 3 1) : 9 меньше или равно 9 с головой правила 1 : X leq X]
(1 3 1) solved : 9 leq 9
[(1 3 1) результат сопоставления: 9 меньше или равно 9]
(2 3 1) : 23 leq 22 trace ? y
[(2 3 1) : 23 меньше или равно 22 трассировка ? да]
matching (2 3 1) : 23 leq 22 with head of 1 : X leq X
[сопоставление (2 3 1) : 23 меньше или равно 22 с головой правила 1 : X меньше или равно X]
match fails
[сопоставление оканчивается неудачей]
matching (2 3 1) : 23 leq 22 with head of 2 : X leq Y
[сопоставление (2 3 1) : 23 меньше или равно 22 с головой правила 2 : X меньше или равно Y]
match succeeds 23 leq 22
[сопоставление успешно завершено : 23 leq 22]
new query : 23 LESS 22
[новый запрос : 23 LESS 22]
(1 2 3 1) : 23 LESS 22
```

* Или, что то же самое, часть подцелей удаётся сопоставить с утверждениями программы. — Прим. пер.

```

[(1 2 3 1) : 23 LESS 22]
(1 2 3 1) fails : 23 LESS 22
[(1 2 3 1) неудача : 23 LESS 22]
retrying (2 3 1)
[снова попробовать (2 3 1)]
(2 3 1) failing : 23 leq 22
[(2 3 1) неудача : 23 меньше или равно 22]
retrying (1 3 1) : 9 leq 9 with head of 2 : X leq Y
[снова попробовать (1 3 1) : 9 меньше или равно 9 с головой
правила 2 : X меньше или равно Y]
match succeeds : 9 leq 9
[сопоставление успешно завершено : 9 меньше или равно 9]
new query 9 LESS 9
[новый запрос 9 LESS 9]
(1 1 3 1) : 9 LESS 9
[(1 1 3 1) : 9 LESS 9]
(1 1 3 1) failing : 9 LESS 9
[(1 1 3 1) неудача : 9 LESS 9]
retrying (131)
[снова попробовать (1 3 1)]
(1 3 1) failing : 9 leq 9
[(1 3 1) неудача : 9 меньше или равно 9]
(3 1) failing : (either 9 LESS 9... etc...)
[(3 1) неудача : (либо 9 LESS 9... и т. д. ...)]
(2 1) failing : date now (X Y Z)
[(2 1) неудача : текущая дата (X Y Z)]
retrying (1 1)...
[снова попробовать (1 1)...]



```

Может показаться, что хороший механизм обработки запросов в данном случае приносит больше вреда, чем пользы. Ведь программа вынуждена оценивать каждое условие в конструкции (either... or...); причем все процессы сопоставления оканчиваются неудачей. После этого управление вновь передается первому утверждению в первом условии (1 1), т. е. делается попытка найти новое значение X, которое содержится в утверждении типа

X born (Y Z x)

С помощью первого правила отношения age возраст Джо определить не удастся. И то, что сопоставление некоторых подцелей в процессе использования этого правила будет успешным, только приведет к лишним затратам времени. После этого программа будет осуществлять поиск возраста Билла и Изабеллы. Для этого потребуются произвести минимальное число возвратов, поскольку решения будут найдены с помощью первого правила.

Таким образом найдена вторая наиболее важная причина неэффективной работы программы. Она заключается в следующем: второе правило отношения age применяется не только к Джо, но

Номера утверждений	Состояние стека	Комментарии
(11)	born 1	Бетти родилась (2 5 1950)
(21)	now 1 born 1	Текущая дата (22 9 1985)
(131)	LESS now 1 born 1	5 < 9
(231)	 LESS now 1 born 1	Возврата нет
(41)	SUM  LESS now 1 born 1	1985 - 1950 = 35



Состояние стека	POP	Результат
	SUM	Печать Betty 35
LESS now 1 born 1		
 LESS		Отношение "у" препятствует осуществлению возврата
now 1 born 1		
born 1	now 1	Нет альтернативных вариантов
born 2	born 1	
	PUSH born 2	

Рис. 3.3. Изменение состояния стека в процессе реализации механизма возврата

Рис. 3.2. Процесс изменения состояния стека во время обработки запроса

и к трем другим именам тоже, поскольку программе не известно, что решение не может быть найдено с помощью второго правила, если оно обнаружено с помощью первого правила. Другими словами, не учитывается, что правила являются взаимно исключающими.

Для того чтобы преодолеть указанные выше недостатки, можно воспользоваться несколькими приемами. Рассмотрим только два из них.

1. Можно использовать два правила age1 и age2 вместо уже имеющихся и, кроме того, ввести новое третье правило, которое даст возможность выбрать либо age1, либо age2. Это третье правило будет иметь вид

X age Y if
(either X age1 Y and/or X age2 Y).

В данном случае отношение «/» позволяет предотвратить использование age2, если использование age1 завершилось успешно.

2. Можно на основе двух правил отношения age сконструировать одно. Причем сделать это надо таким образом, чтобы кандидат на решение, удовлетворяющий двум первым условиям:

X born (Z x y)

date now (z X1 Y1)

в том случае, если он не удовлетворяет остальным условиям первого правила, сразу же использовался для проверки оставшихся условий второго правила.

Перед тем как идти дальше, наверное, полезно проследить, как меняется содержимое стека в процессе обработки запроса (рис. 3.2 и 3.3). Изображенное на этих рисунках не следует интерпретировать как точную копию состояний памяти — скорее рисунки дают обобщенную картину выполняемых процессов. Центральный столбец на рис. 3.2 отражает структуру стека на разных этапах обработки запроса; левый столбец содержит составные номера утверждений, используемые и программой трассировки; и, наконец, правый столбец содержит информацию о последнем помещенном в стек элементе. Отметим, что, анализируя рисунки, необходимо учитывать существование специальных внутренних кодов, используемых для представления различных элементов, помещенных в стек.

Элементы попадают в стек именно в том порядке, в котором они обрабатываются программой; поэтому первым, помещенным в стек, элементом является

born 1

Это обозначает не что иное, как первое утверждение отношения born, которое в конце концов будет интерпретироваться как Betty born (2 5 1950)

Затем в стек будет помещен элемент now 1 — первое утверждение (оно одновременно и единственное) отношения now. После него — 5 LESS 9, затем символ «/», используемый для предотвращения возврата при поиске альтернативных вариантов, и, наконец, будет получено решение 35 = 1985 — 1950. Заключительное состояние стека перед возвратом изображено в нижней части рис. 3.2. Именно последний помещенный в стек элемент обеспечивает ответ на запрос, который будет выглядеть так:

Betty age 35

На рис. 3.3 показано изменение состояний стека в результате операций, выполняемых в процессе возврата. Последний элемент, представляющий собой отношение SUM, выталкивается из стека и повторно не обрабатывается, поскольку альтернативных решений для него нет. Если бы альтернативное решение существовало, оно помешалось бы в стек и печаталось. Следующим выталкиваемым из стека элементом является условие (either ... or ...), но присутствие отношения «/» предотвращает на данном шаге выполнение возврата для поиска альтернативных вариантов. В конце концов из стека выталкивается born 1 и следом за

этим в стек сразу же помещается born 2. Далее повторяется процесс, аналогичный изображенному на рис. 3.2, и так продолжается до тех пор, пока все утверждения, описывающие отношение born, не будут исчерпаны. В результате использования первого утверждения отношения age никаких новых решений, кроме

Betty born (2 5 1950)

получено не будет. После этого аналогичная процедура повторяется для второго утверждения отношения age.

Заметим, что с помощью программы трассировки пользователю удастся получить очень много различной информации.

В связи с этим можно рекомендовать начинающим использовать ее для своих собственных преимущественно небольших программ, написанных на Прологе. Ведь хорошая техника программирования может быть приобретена лишь при условии понимания причин, обуславливающих неэффективность работы программы.

Ниже приведена более эффективная программа, позволяющая определить возраст по дате рождения и текущей дате:

Betty born (3 5 1950)

Joe born (23 9 1917)

Bill born (22 9 1943)

Isabel born (21 9 1942)

X leq X

X leq Y if

X LESS Y

X age Y if

X born (Z x y) and

(either (either x LESS 9 and / or x leq 9 and Z leq 22 and /) and
SUM (Y y 1985) and / or SUM (Y y 1984))

Возможность создания такой программы появилась только после изучения протокола трассировки первоначальной версии программы, выполняющей ту же самую функцию. Отметим главные преимущества последнего варианта программы. Во-первых, отношение age определяется теперь с помощью только одного утверждения; во-вторых, следует определять текущую дату и, в-третьих, те люди, чьи дни рождения в текущем году еще не прошли, распознаются автоматически, поскольку данные о них не удовлетворяют первой части правила. Информацию о текущей дате можно включить в правило с помощью отношения update, использующего, в свою очередь, отношение is-told. Но если Вам необходимо провести трассировку, то лучше все же избежать хотя бы временного применения отношения update, что позволит сэкономить память. Ниже приводится часть протокола трассировки последней версии программы:

all-trace (x y: x age y)
 [полная трассировка (x y: x имеет возраст y)]
 (1) : X age Y trace ? y
 [(1) : X имеет возраст Y трассировка ? да]
 matching (1) : X age Y with head of 1 : Z age x
 [сопоставление (1) : X имеет возраст Y с головой правила
 (1) : Z имеет возраст x]
 match succeeds : X age Y
 [согласование успешно завершено : X имеет возраст Y]
 new query : X born (Y Z x) and (either (either Z LESS 9 and/or
 Z leq 9
 [новый запрос : X родился (Y Z x) и (либо (либо Z LESS 9
 и/либо Z меньше или равно 9]
 and Y leq 22 and/) and SUM (y x 1985) and/or SUM (y x 1984))
 [и Y меньше или равно 22 и /) и SUM (y x 1985) и/либо (y x
 1984))]
 (1 1) : X born (Y Z x) trace ? y
 [(1 1) : X родился (Y Z x) трассировка ? да]
 matching (1 1) : X born (Y Z x) with head of 1 : Betty born
 (3 5 1950)
 [сопоставление (1 1) : X родился (Y Z x) с головой правила
 1 : Бетти родилась (3 5 1950)]
 match succeeds : Betty born (3 5 1950)
 [сопоставление успешно завершено : Бетти родилась (3 5 1950)]
 (1 1) solved : Betty born (3 5 1950)
 [(1 1) результат сопоставления : Бетти родилась (3 5 1950)]
 (2 1) : (either (either 5 LESS 9 and/or 5 leq 9 and 3 leq 22 and /)
 and
 [(2 1) : (либо (либо 5 LESS 9 и / либо 5 меньше или равно 9
 и 3 меньше или равно 22 и /) и]
 SUM (X 1950 1985) and/or SUM (X 1950 1984)) trace? (y/n) y
 [SUM (X 1950 1985) и / либо SUM (X 1950 1894)) трассировка ?
 (да/нет) да]
 (2 1) either branch
 [(2 1) ветвление либо-либо]
 (1 2 1) : (either 5 LESS 9 and/or 5 leq 9 and 3 leq 22 and/) trace ?
 (y/n) y
 [(1 2 1) : (либо 5 LESS 9 и/либо 5 меньше или равно 9 и 3
 меньше или равно 22 и /) трассировка ? (да/нет) да]
 (1 2 1) either branch
 [(1 2 1) ветвление либо-либо]
 (1 1 2 1) : 5 LESS 9
 [(1 1 2 1) : 5 LESS 9]
 (1 1 2 1) solved : 5 LESS 9
 [(1 1 2 1) результат сопоставления : 5 LESS 9]
 (2 1 2 1) : /
 [(2 1 2 1) : /]
 (2 1 2 1) solved : /


```

[(2 1 2 1) результат сопоставления : /]
(1 2 1) solved : (either 5 LESS 9 and/or 5 leq 9 and 3 leq 22
and/)
[(1 2 1) результат сопоставления : либо 5 LESS 9 и/либо 5
меньше или равно 9 и 3 меньше или равно 22 и/]
(2 2 1) : SUM (X 1950 1985)
[(2 2 1) : SUM (X 1950 1985)]
(2 2 1) solved : SUM (35 1950 1985)
[(2 2 1) результат сопоставления : SUM (35 1950 1985)]
(3 2 1) : /
[(3 2 1) : /]
(3 2 1) solved : /
[(3 2 1) результат сопоставления : / ]
(2 1 solved : (either (either 5 LESS 9 and/or 5 leq 9 and 3 leq 22
and/)
[(2 1) результат сопоставления : (либо/либо 5 LESS 9 и/либо 5
меньше или равно 9 и 3 меньше или равно 22 и/)]
(1) solved : Betty age 35
[(1) результат сопоставления : Бетти имеет возраст 35]
Betty 35
[Бетти]
backtracking ...
[возврат для поиска альтернативных вариантов ...]
(2 1) failing (either (either 5 LESS 9 and/or 5 leq 9 and 3 leq
22 and/)
[(2 1) неудача : (либо (либо 5 LESS 9 и/либо 5 меньше или
равно 9 и 3 меньше или равно 22 и/)]
and SUM (X 1950 1985) and/or SUM (X 1950 1984))
[и SUM (X 1950 1985)) и/либо SUM (X 1950 1984))]
retrying (1 1)
[снова попробовать (1 1)]

```

Далее приводится текст отношения update, использование которого позволяет пользователю вводить нужную ему дату:

```

update X if
  age KILL and
  (date now (Y Z x) is-told) and
  SUM (y 1 x) and
  (z age X1 if z born (Y1 Z1 x1) and (either (either Z1 LESS Z and /
or Z1 leq Z and Y1 leq Y and /) and SUM (X1 x1 x) and / or SUM
(X1 x1 y))) add

```

Чтобы использовать отношение update, необходимо ввести
all (x: update x)
[определить все (x : модифицировать x)]

Отметим, что x — это фиктивная переменная, поэтому в результате обработки запроса никакого ответа получено не будет.

Но зато данные, составляющие новую дату, будут включены в отношение age.

Анализ протокола трассировки позволяет заключить, что попытки осуществить возврат с целью поиска альтернативных решений в соответствии с конструкцией (either ... or ...) сразу игнорируются из-за присутствия отношения «/».

3.7. РЕКУРСИВНЫЕ ОПРЕДЕЛЕНИЯ

Как уже отмечалось, рекурсия является одним из средств описания отношений. Недостаток, свойственный рекурсивным определениям, заключается в том, что в процессе обработки запроса размер стека может расти очень быстро, поскольку предусмотрено сохранение частичных решений. Наиболее часто для иллюстрации рекурсивного отношения используется отношение, позволяющее вычислить факториал числа. Факториал определяется так:

$$N! = (N) (N - 1) (N - 2) \dots (3) (2) (1)$$

т. е. $4! = 4 \times 3 \times 2 \times 1 = 24$. Факториал 1 равен 1. Теперь можно сформировать отношения для вычисления факториала

```
1 fact 1
X fact Y if
    1 LESS X and
    SUM (Z 1 X) and
    Z fact x and
    TIMES (X x Y)
```

Для проверки работы правил, описывающих это отношение, можно использовать следующий тест:

```
all (x: 4 fact x)
[определить все (x: 4 fact x)]
24
[24]
No (more) answers
[Ответов (больше) нет]
```

Ниже приведен протокол работы программы *, получаемый в ответ на запрос:

```
all = trace (x: 3 fact x)
```

Отметим только, что в нем опущены строки, в которых сообщается об успешном сопоставлении переменных. Это сделано не столько с целью экономии места, сколько для того, чтобы попытаться скон-

* Этот протокол не снабжается переводом, поскольку, если читатель внимательно изучил предыдущие протоколы трассировок, он без труда разберется и в этом. — *Прим. пер.*

центрировать внимание читателей на операциях во стеком и процессах возврата.

```
(1 1): 1 LESS X
(1 1) solved: 1 LESS 3
(2 1) SUM (X 1 3)
(2 1) solved: SUM (2 1 3)
...
(1 3 1): 1 LESS 2
(1 3 1) solved: 1 LESS 2
(2 3 1): SUM (X 1 2)
(2 3 1) solved: SUM (1 1 2)
...
(3 3 1) solved: 1 fact 1
(4 3 1): TIMES (2 1 X)
(4 3 1) solved: TIMES (2 1 2)
(3 1) solved 2 fact 2
(4 1) TIMES (3 2 X)
(4 1) solved: TIMES (3 2 6)
(1) solved: 3 fact 6
backtracking ...
(4 1) failing ...
(4 3 1) failing ...
(1 3 3 1): 1 LESS 1
(1 3 3 1) failing: 1 LESS 1
retrying (3 3 1)
(3 3 1) failing: 1 fact X
(2 3 1) failing: SUM (X 1 2)
(1 3 1) failing: 1 LESS 2
retrying (3 1)
(3 1) failing: 2 fact X
(2 1) failing: SUM (X 1 3)
(1 1) failing: 1 LESS 3
retrying (1)
(1) failing: 3 fact X
No (more) answers
```

В процессе вычисления $N!$ отношения LESS и SUM используются для каждого из чисел: N , $N-1$, $N-2$, ..., 2, 1. Результаты сопоставления каждый раз помещаются в стек, хотя в данном случае никаких альтернативных вариантов сопоставления заведомо не существует. N раз используется и отношение TIMES. Результаты его работы также попадают в стек. Ясно, что в процессе вычисления факториала для больших чисел размер стека будет расти очень быстро. Кроме того, значительное время будет тратиться на реализацию возвратов. Таким образом, можно сказать, что для работы с рекурсивными определениями отношений, аналогичными определенному выше, потребуется память, объем которой пропорционален числу обращений к рекурсивному правилу. В данном случае для вычисления факториала потребуется память, объем которой пропорционален N , где N — число, факториал которого определяется.

3 fact 6
TIMES (326)
2 fact 2
TIMES (212)
1 fact 1
SUM (112)
1 LESS 2
SUM (213)
1 LESS 3

Рис. 3.4. Процесс изменения состояния стека во время вычисления факториала числа 3

Отметим, что у читателя может вызвать удивление тот факт, что рядом с условиями (1 1), (2 1), (3 1) и другими после осуществления возврата появляется сообщение: «failing» (неудача), в то время как ранее выполнение этих условий завершалось успешно. Дело в данном случае в том, что при осуществлении возврата Пролог-система старается найти решения, отличные от тех, которые привели к успешному сопоставлению.

На рис. 3.4 показано содержимое стека в процессе вычисления факториала числа 3. Механизм возврата начнет работать именно тогда, когда число элементов в стеке достигнет девяти. Все элементы по очереди будут выталкиваться из стека, поскольку любые попытки поиска альтернативных вариантов обречены на неудачу.

Но если стек растет так быстро при вычислении всего-навсего 3!, то размеры его станут поистине гигантскими при определении факториалов больших чисел.

Мы уже рассказывали читателям об отношении «/». Напомним, что оно может использоваться для того, чтобы программа могла считать условия, сопоставление которых прошло успешно, не подлежащими дальнейшей проверке, т. е. для того, чтобы предотвращать возвраты с целью поиска альтернативных вариантов. Такая процедура носит название «выталкивания в случае успеха». Ее реализация приводит к уменьшению размера стека за счет удаления из него элементов, сопоставление которых было завершено успешно. Рекурсия специального типа, которая называется хвостовой *, также позволяет ограничить рост стека и строго контролировать процесс возврата. Это происходит благодаря очистке стека после успешного сопоставления условия, содержащего рекурсию.

Как известно, любое рекурсивное определение содержит по крайней мере одно нерекурсивное правило и одно или несколько правил с рекурсией. В большинстве случаев в определении имеется по одному правилу каждого типа. Считается, что используется хвостовая рекурсия, если последнее условие в последнем правиле является рекурсивным. Данное выше определение факториала является рекурсивным, поскольку во втором правиле используется отношение fact. Но это не хвостовая рекурсия, так как отношение fact не является последним в последнем (в данном случае во втором) правиле. Приведем пример определения отношения, использующего хвостовую рекурсию:

* Новый термин — хвостовая рекурсия (tail recursion). — *Прим. пер.*

$X \text{ in } (X Y)$
 $X \text{ in } (Y Z) \text{ if}$
 $\text{SUM } (Y \ 1 \ x) \text{ and}$
 $x \text{ LESS } Z \text{ and}$
 $X \text{ in } (x \ Z)$

Это отношение можно использовать для генерации последовательности чисел $X, X + 1, X + 2, \dots, Z - 1$. Аналогичная задача решалась в гл. 2 с помощью других отношений. В данном случае рекурсия содержится в последнем условии последнего правила. Ниже дан протокол трассировки

```

all-trace (x: x in (1 3))
(1): X in (1 3) trace ?y
matching (1): X in (1 3) with head of 1: Y in (Y Z)
match succeeds: 1 in (1 3)
(1) solved: 1 in (1 3)
1
backtracking ...
matching (1): X in (1 3) with head of 2: Y in (Z x)
match succeeds: X in (1 3)
new query: SUM (1 1 X) and X LESS 3 and Y in (X 3)
(1 1): SUM (1 1 X)
(1 1) solved: SUM (1 1 2)
(2 1): 2 LESS 3
(2 1) solved: 2 LESS 3
(3 1): X in (2 3) trace ?y
matching (3 1): X in (2 3) with head of 1: Y in (Y Z)
match succeeds: 2 in (2 3)
(3 1) solved: 2 in (2 3)
matching (3 1): X in (2 3) with head of 1: Y in (Y Z)
match succeeds: 2 in (2 3)
(3 1) solved: 2 in (2 3)
(1) solved: 2 in (1 3)
backtracking ...
retrying (3 1)
matching (3 1): X in (2 3) with head of 2: Y in (Z x)
match succeeds: X in (2 3)
new query: SUM (2 1 X) and X LESS 3 and Y in (X 3)
(1 3 1): SUM (2 1 X)
(1 3 1) solved: SUM (2 1 3)
(2 3 1): 3 LESS 3
(2 3 1) failing: 3 LESS 3
(1 3 1) failing: SUM (2 1 X)
retrying (3 1)
(3 1) failing: X in (2 3)
(2 1) failing: 2 LESS 3
(1 1) failing SUM (1 1 X)
retrying (1)
(1) failing: X in (1 3)
No (more) answers
  
```

2 in (1 3)
2 in (2 3)
2 LESS 3
SUM(112)
1 in (13)

Рис. 3.5. Состояние стека в процессе обработки запроса для программы с хвостовой рекурсией

На рис. 3.5 изображено состояние стека во время обработки запроса. Попробуем на этом примере разобраться, в чем сила хвостовой рекурсии. Заметим, что в процессе оценки каждого числа используются три отношения — SUM, LESS и in. После того как сопоставление цели с правилом 2 успешно проведено, стек очищается и программа переходит к поиску следующего по порядку числа * до тех пор, пока сопоставление не закончится неудачей.

Упражнение 3.12

Составьте содержащее хвостовую рекурсию отношение для получения расположенной в порядке убывания последовательности целых чисел в интервале (X Y), причем X должно входить в эту последовательность. Для получения протокола трассировки типичного запроса используйте программу SIMTRACE.

Упражнение 3.13

Возрастающую последовательность можно получить, изменив на противоположный порядок следование правил, описывающих отношение in. Можно ли считать, что это новое определение содержит хвостовую рекурсию. Проконтролируйте работу новой программы с помощью SIMTRACE.

Значительный эффект может принести использование отношений с хвостовой рекурсией в работе со списками. В приложении книги «Руководство по микроПрологу» приводится пример отношения APPEND, содержащего хвостовую рекурсию:

```
append (( ) X X)
append ((X Y) Z (X x) if
      append (Y Z x)
```

В руководстве отмечается, что размер стека не зависит от длины обрабатываемого списка.

3.8. ХВОСТОВАЯ РЕКУРСИЯ И СПИСКИ

В этом разделе методы хвостовой рекурсии будут использованы в программах, предназначенных для хранения и обработки данных о людях (см. гл. 2 и 3). Предположим, что необходимо хранить и уметь организовывать доступ к данным о людях, которые

* Можно добавить, что в стеке в любой момент времени находятся не более двух отношений SUM и LESS. — *Прим. пер.*

Таблица 3.3

Клиент	Car ins. (Страховка за автомобиль)	Car tax (Налог на автомобиль)	Life ins. (Личная страховка)	Home ins. (Страховка на здание)	TV (Лицензия на ТВ)	Club (Взнос за членство)
Alan (Алан)	(20 3)	(9 6)	(20 3)	0	0	0
Betty (Бетти)	(15 8)	(24 9)	(6 10)	(30 7)	(1 11)	(4 5)
Colin (Колин)	0	0	0	0	(22 11)	(5 9)

ежегодно оплачивают следующие счета: страховку за автомобиль, личную страховку, страховку за здание, лицензию на право пользования телевизором, налог на автомобиль и взносы за членство в клубе или какой-то профессиональной ассоциации. Любой человек может иметь счет по конкретной позиции, а может и не иметь. Нас будут интересовать месяц и число погашения счета и не будут — год и размер счета. Представляем все эти данные в виде табл. 3.3.

Отсутствие информации между скобками говорит о том, что данному человеку по данной позиции счет не выставляется. Отметим, что в дальнейшем в случае необходимости данные, содержащиеся в табл. 3.3, могут быть изменены. Легко организовать хранение всех данных в следующей форме:

Alan data ((20 3) (9 6) (2 0 3) () () ())
 Betty data ((15 8) (24 9) (6 10) (30 7) (1 11) (4 5))
 Colin data (() () () () (22 11) (5 9))

Допустим, что необходимо уметь извлекать из программы следующую информацию:

1. Каковы имена людей, которые погасили все счета, и какова дата оплаты?
2. Какие счета погашены на текущий день?
3. Кто погасил счет данного типа и какова дата оплаты?
4. Кто не погасил счет данного типа?
5. Кого необходимо предупредить о том, что счет по заданной позиции должен быть погашен в течение определенного числа дней?

Ранее уже были даны оценки тем методам, которые можно положить в основу организации программы, позволяющей ответить на аналогичные запросы. Например, в данном случае можно сформировать для счета каждого типа отдельные отношения или использовать для поиска данных процесс сопоставления со списком данных в отношении data. Последний метод можно реализовать с помощью отношений типа

X car ins Y if
 X data (Y | Z)

Идея еще одного метода заключается в помещении списка типов счетов в унарное отношение

bill ((car ins) (car tax) (life ins) (home ins) (tv) (club))

Теперь необходимо связать тип счета с именем человека и датой оплаты. Это можно сделать, если удастся, в свою очередь, связать аргумент X списка счетов с аргументом списка данных определенной личности. Следующее отношение, использующее хвостовую рекурсию, позволяет связать аргументы, стоящие в списках на одних и тех же местах:

$$\begin{aligned} (X \ Y) \text{ match } ((X \mid Z) \ (Y \mid x)) \\ (X \ Y) \text{ match } ((z \mid x) \ (y \mid z)) \text{ if} \\ \quad (X \ Y) \text{ match } (x \ z) \end{aligned}$$

Здесь первое нерекурсивное правило позволяет связать головы списков. Второе правило дает возможность связать между собой элементы списков, стоящие на одних и тех же местах. Определение отношения содержит хвостовую рекурсию, поскольку рекурсивное условие стоит последним в последнем правиле. С помощью отношения match (связать) можно получить множество упорядоченных пар аргументов; первым аргументом в каждой паре является член первого списка, вторым аргументом — связанный с членом первого списка член второго списка. Для проверки работы отношения используется следующий тест:

```
all (x y : Alan data z and bill X and (x y) match (z X))
[определить все (x y : Алан, данные z и счет X и (x y) связать
(z X)]
(20 3) (car ins)
[(20 3) (страховка за автомобиль)]
(9 6) (car tax)
[(9 6) (налог на автомобиль)]
(20 3) (life ins)
[(20 3) (личная страховка)]
( ) (home ins)
[( ) (страховка за здание)]
( ) (tv)
[( ) (лицензия на ТВ)]
( ) (club)
[( ) (взнос за членство в клубе)]
```

В данном примере в виде упорядоченных пар печатаются даты оплат, произведенных Аланом, и связанные с ними названия счетов из списка счетов.

Упражнение 3.14

а. Составьте запрос, позволяющий определить имена вместе с названиями счетов и датами оплат.

б. Составьте запрос, аналогичный запросу, описанному в п. (а). Отличие заключается в том, что информация о счетах, по которым данный человек не платит, выводиться не должна.

Чтобы выполнить упражнение 3.14, а, можно использовать новое отношение pays (оплачивает):

```
X pays (Y on Z) if
    bill x and
    X data y and
    (Y Z) match (x y)
```

и запрос

```
all (x y z: x pays (y on z))
```

В ответ будет получено:

```
Alan (car ins) (20 3)
Alan (car tax) (9 6)
Alan (life ins) (20 3)
```

и т. д.

Если не надо печатать названий счетов, по которым данный человек не производит оплат, нужно следующим образом изменить отношение pays:

```
X pays (Y on Z) if
    bill x and
    X data y and
    (Y Z) match (x y) and
    not (Y ( )) match (x y)
```

т. е. в данном случае пустой список () исключается из процесса сопоставления. Ниже приведен еще один вариант отношения, использование которого позволяет решить ту же задачу:

```
X pays (Y on Z) if
    bill x and
    X data y and
    (Y Z) match (x y) and
    not Z EQ ( )
```

В последнем варианте фигурирует отношение EQ. Напомним, что это отношение истинно в том случае, когда два аргумента, представляющие собой списки, полностью (включая порядок следования элементов) идентичны.

Упражнение 3.15

Добавьте к программе отношение, позволяющее задавать текущую дату. После этого составьте отношения для определения счетов:

а) погашение которых произведено только что;

- б) оплата по которым будет произведена в следующем месяце;
- в) оплаты по которым будут произведены в оставшиеся дни текущего месяца.

Упражнение 3.16

Покажите, как отношение `match` можно использовать вместе с отношением `stat` из разд. 3.3 для извлечения, во-первых, подсписков из основного списка и, во-вторых, информации из нужного подподписка. В данном случае дело облегчается тем, что каждый подподписок содержит три элемента. Предусмотрите случай, когда подподписки будут включать различное число элементов.

Прочитав эту главу, читатели должны понять, что список, или структура данных является довольно мощным средством программирования. Одно из достоинств Пролога заключается в том, что достаточно нескольких простых примеров для того, чтобы показать, как могут создаваться очень большие программы. Пролог-система позволяет добавлять столько дополнительных данных или дополнительных правил, сколько можно разместить в оперативной памяти. Кроме того, одну и ту же программу, написанную на Прологе, можно использовать для решения многих различных задач. Хотя уже было описано довольно много методов, позволяющих сопоставлять элементы, принадлежащие различным спискам, читателям не составит особого труда придумать свои собственные.

Долг программиста состоит в том, чтобы уметь создавать удобные для сопровождения хорошо документированные программы, легко поддающиеся расширению и удовлетворяющие требованиям гибкости. Кроме того, не надо забывать, что программист постоянно должен заботиться о скорости работы программы и об экономном расходовании памяти. В связи с этим обратим внимание на использование оператора `«/»`. Этот оператор позволяет за счет отсечения альтернативных вариантов эффективно управлять возвратом. Аналогично метод, основанный на применении хвостовой рекурсии при задании отношений, дает возможность ограничить рост стека. Настоятельно рекомендуем читателям использовать именно этот метод. В качестве образцов можно взять стандартные отношения микроПролога; для того чтобы посмотреть правила, определяющие, например, отношения `ON` и `APPEND`, достаточно ввести команды

```
LIST ON
LIST APPEND
```

Кроме того, можно распечатать тексты любых модулей, входящих в состав той или иной программной системы. Так, сервисная программа `SIMPLE` состоит из трех модулей: `query-mod`, `er-gmess-mod`, `program-mod`. Текст любого из них можно вывести

на экран с помощью команды LIST. Напомним, что просмотр длинных программ можно приостановить нажатием клавиш SYMBOL SHIFT и A.

Ответы к упражнениям

Упражнение 3.1

- а) 1, (2 4), (), a2, 2, a;
- б) ab, abc, (a (Joe Ted) John), def

Упражнение 3.2

- а) (THIS IS A LIST); б) (WHAT); в) (IS); г) ab (отметим, что ab — один терм); д) ab (cd).

Упражнение 3.3

- а) all (x y: x stat y)
- б) which (x: Bill job x)
- в) all (x y z: x aged X and 30 LESS X and x job y and x earns z)
- г) all (x: Frances stat x)
- д) см. следующее упражнение.

Упражнение 3.4

X hobby Y if

X stat (Z x y z X1 Y1 Y Z1)

X car Y if

X stat (Z x y z X1 Y1 Z1 Y X2)

X music Y if

X stat (Z x y X1 X1 Z1 X2 Y Y2)

Bill stat (42 butcher 12000 23 19 7 golf Volvo jazz) ...

Упражнение 3.5

См. следующее упражнение.

Упражнение 3.6

а. all (X Y: X marriage (Z x y) and Y marriage (z X1 y) and X LESS Y)
В ответ на этот запрос должно быть получено: Helen, Ted.

б. all (X Y: X work (Z x y) and Y work (z X1 y) and X LESS Y)
В ответ на этот запрос должно быть получено: Helen, Tom.

в. all (X: X marriage (Y Z x) and X work (y z x)) Ответ — Betty. Действительно, Бетти — единственная, чей стаж совпадает с числом лет, прожитых в браке.

г. all (X: X birth (Y Z x) and X home (y z x))

д. all (X: X birth (Y Z x) and X home (y z x) and X work (X1 Y1 x))

Упражнение 3.7

Запрос, приведенный в упражнении, позволяет определить тех, кто работает не там, где живет, и, кроме того, тех, кто вообще не работает. Правильный запрос можно получить, поменяв порядок следования целевых утверждений

all (x y: x workplace y and not x homeplace y)

Упражнение 3.8

а) all (x y: x earns y and 8000 LESS y)

б) all (x: x aged y and x earns z and 30 LESS y and z LESS 10000)

- в) is (x home-no y and x work-no y)
 г) all (x: x years-in-job y and Tom years-in-job z and z LESS y)
 д) all (x: x married y and joe married z and z LESS y)

Упражнение 3.9

- а) all (x y: x age y and Ted age z and y LESS z)
 б) считается, что в программе присутствует утверждение

date now (1 1 1975)
 all (xy: x age y and 18 LESS y)

- в) считается, что в программу включено утверждение

date now (31 12 2000)
 all (xy: x age y and y LESS 50)

- г) all (xy: x age z and y age z and x LESS y)
 д) all (xy: x age z and y age z and x LESS y and x born X and y born X)
 е) проще всего уничтожить имеющуюся дату, ввести

date now (1 1 1950)

и сформировать вопрос следующего вида:

all (x: x age y and y LESS 0)

Упражнение 3.10

McDoo stock (30 12 200 555)
 Fee stock (598 100 3500 9760)
 Putlog stock (8900 950 7380 6875)
 X sand Y if
 X stock (Y|Z)
 X cement Y if
 X stock (Z Y|x)
 X blocks Y if
 X stock (Z x Y|y)
 X bricks Y if
 X stock (Z x y Y|z)
 X newstock (Y Z x y) if
 X stock (z X1 Y1 Z1) and
 (X adjust (x1 y1 z1 X2) is-told and
 SUM (Y x1 z) and
 SUM (Z y1 X1) and
 SUM (x z1 Y1) and
 SUM (y X2 Z1) and
 (X stock Y2) delete and
 (X stock (Y Z x y)) add and /

Упражнение 3.11

- а) X age Y if
 A upcount and ...
 X age Y if
 B upcount and ...
 б) all (xy zX: x age y and A count z B count X)

в) данные о возрасте людей печатаются в первую очередь, поскольку они определяются в соответствии с первым правилом.

Упражнение 3.12

X in (Y X)
X in (Y Z) if
SUM (X 1 Z) and
Y LESS x and
X in (Y x)

Упражнение 3.13

Определение не содержит хвостовой рекурсии, поскольку последнее правило не является рекурсивным.

Упражнение 3.14

- a) all (x y z: x pays (y on z))
б) можно использовать либо

all (x y z: pays (y on z) and not z EQ ())

либо

all (x y z: paus (y on z) and not x pays (y on ()))

Упражнение 3.15

Необходимо ввести в программу отношение

date now (2 0 3)

- a) X due y if
x pays (y on z) and
date now z
б) X next-month Y if
X pays (Y on (Z x)) and
date now (y z) and
SUM (z 1 x)
в) X this-month if
X pays (Y on (Z x)) and
date now (y x) and
y LESS

Определить истинность логического выражения, т. е. выяснить, какое значение оно принимает — истина или ложь, можно с помощью математического аппарата, называемого исчислением высказываний. Результатом развития этого аппарата явилось исчисление предикатов, положенное в основу механизма логического вывода языка Пролог. Для представления логических выражений как в исчислении высказываний, так и в исчислении предикатов применяется символьная запись, которая также удобна и при отработке задачи на ЭВМ.

В вычислительной технике термин «логика», часто используется применительно к электронным схемам, преобразующим сигналы нескольких (обычно двух) уровней. Но мы подразумеваем под этим термином понятие, которое больше соответствует человеческому способу рассуждений, чем функционированию электронных схем.

Объектами исчисления высказываний служат обычные предложения, например: «Сегодня падал снег», «Джо любит жареную рыбу», «Изучить логику просто» и т. д. Об этих предложениях всегда можно сказать, что они истинны или ложны, т. е. принимают значения «истина» или «ложь». Предположим, имеется высказывание: «Джо любит жареную рыбу». Если Джо действительно предпочитает это блюдо другим, то приведенное высказывание принимает значение истины, в противном случае — лжи. Очевидно, что предложения типа «Сколько времени?» или «Проклятая логика» не могут быть объектами исчисления высказываний.

Истину и ложь в символьной записи принято обозначать соответственно Т (от true — истина) и F (от false — ложь). В настоящее время существует еще один тип логического исчисления, называемый нечеткой логикой. В этом исчислении можно оперировать с оценками степени уверенности в истинности высказываний. Так, например, для высказывания «Сегодня будет дождь» может быть определена вероятность истинности, равная 55%. В дальнейшем указанная вероятность может быть использована в вычислениях. Нечеткая логика сейчас широко используется в системах ИИ, а особенно в такой его ветви, как экспертные системы. Однако описание этого вида логики выходит за пределы данной

книги. В обычном исчислении высказываний могут быть заданы предложения типа «С вероятностью 55% сегодня будет дождь», которые могут принимать одно из двух значений — истина или ложь, но какая-либо другая оценка таких высказываний в данном исчислении невозможна.

4.1. КОМБИНАЦИИ ВЫСКАЗЫВАНИЙ

Два любых высказывания можно объединить в одно с помощью двух основных операций: конъюнкции — логического И, обозначаемого AND, и дизъюнкции — логического ИЛИ, обозначаемого OR. Так, например, высказывания «Вчера было пасмурно» и «Сегодня утром шел снег» можно связать конъюнкцией и получить высказывание «Вчера было пасмурно и сегодня утром шел снег». Из тех же высказываний с помощью дизъюнкции можно получить следующее: «Вчера было пасмурно или сегодня утром шел снег». Высказывание, полученное с помощью конъюнкции, принимает значение истина только тогда, когда значения обоих входящих в него высказываний истинны. Во всех остальных случаях конъюнкция ложна. Дизъюнкция принимает значение лжи только тогда, когда оба входящих в нее высказывания ложны. В остальных случаях дизъюнкция равна истине.

В логике используется также другая операция, напоминающая дизъюнкцию, которая называется ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR). Высказывание, получаемое с помощью данной операции, принимает значение истина при условии, что одно, и только одно, из входящих в него высказываний имеет значение истинны. В разговорной речи такие высказывания часто объединяются конструкцией «или ..., или ...», например: «Или я пойду на футбольный матч, или будут смотреть его прямую трансляцию по телевизору». Очевидно, что истинность одной составляющей исключает истинность другой.

Другая важная операция, используемая в логике, — это отрицание, логическое НЕ, обозначаемое NOT. Например, из высказывания «Сегодня идет снег» можно с помощью отрицания получить «Сегодня не идет снег». Представление значений слож-

Таблица 4.1

Истинность операций AND (И), OR (ИЛИ), XOR (Исключающее ИЛИ) и NOT (НЕ)

Q	P	P AND Q	P OR Q	P XOR Q	P	NOT P
T	T	T	T	F	T	F
T	F	F	T	T	F	T
F	T	F	T	T	—	—
F	F	F	F	F	—	—

Противоречие и тавтология

P	NOT P	P AND NOT P	P OR NOT P
T	F	F	T
F	T	F	T

ных высказываний для каждой возможной комбинации значений их составляющих осуществляется с помощью так называемых таблиц истинности. Например, в табл. 4.1 одновременно представлены четыре таблицы истинности для операций И (AND), ИЛИ (OR), исключающее ИЛИ (XOR) и НЕ (NOT). В данной таблице буквами P и Q обозначаются любые два логических высказывания.

Рассмотрим логические функции, представленные в табл. 4.2.

Как видно из табл. 4.2, высказывание $P \text{ AND NOT } P$ ($P \text{ И НЕ } P$) всегда принимает значение ложь, тогда как высказывание $P \text{ OR NOT } P$ ($P \text{ ИЛИ НЕ } P$), наоборот, всегда имеет значение истина. Высказывание, всегда имеющее значение истина, называется тавтологией. Высказывание с постоянным значением ложь называется противоречием. Читатель может самостоятельно проверить, что высказывание, составленное с помощью операции XOR из высказываний P и NOT P, представляет собой тавтологию. Другим не менее полезным упражнением будет сравнение таблиц истинности для высказываний $P \text{ XOR } Q$ и $P \text{ AND NOT } Q \text{ OR NOT } P \text{ AND } Q$. Заметим, что следует учитывать приоритеты логических операций: сначала выполняется отрицание (NOT), затем конъюнкция (AND) и, наконец, дизъюнкция (OR). Изменить порядок выполнения операций можно с помощью скобок: действия в скобках производятся в первую очередь. В том, что результат зависит от порядка выполнения операций, можно убедиться, сравнив, например, таблицы истинности для высказываний $\text{NOT } P \text{ AND } Q$ и $\text{NOT } (P \text{ AND } Q)$.

В заключение данного раздела еще раз напомним, что хотя все логические высказывания — это обычные предложения естественного разговорного языка, обратное не всегда верно, т. е. не все предложения естественного языка могут служить логическими высказываниями. Приведем еще один пример. О фразе «Это предложение ложно» нельзя сказать, что она имеет значение истина или ложь. Следовательно, данная фраза не является логическим высказыванием.

4.2. БУЛЕВА АЛГЕБРА

Английский математик Джордж Буль (1815—1864) сформулировал правила для вычисления значений комбинированных логических высказываний и представил их в символической форме. Ос-

$$\text{NOT}(\text{NOT } P) = (P \text{ AND } P) = (P \text{ OR } P) = P$$

P	NOT P	NOT (NOT P)	P AND P	P OR P
T	F	T	T	T
F	F	F	F	F

новными операциями булевой алгебры являются уже знакомые читателю операции AND, OR и NOT от двоичных переменных, обозначаемых буквами латинского алфавита. Из основных свойств указанных операций, иллюстрируемых табл. 4.1 и 4.2, можно вывести следующие три правила:

Двойное отрицание
Идемпотентность
Поглощение

$$\begin{aligned}\text{NOT}(\text{NOT } P) &= P \\ P \text{ AND } P &= P, P \text{ OR } P = P \\ P \text{ AND } (P \text{ OR } Q) &= P\end{aligned}$$

В табл. 4.3 приведены таблицы истинности, позволяющие проверить правила двойного отрицания и идемпотентности. Построение таблицы для проверки правила поглощения мы предоставляем читателям в качестве упражнения.

Одним из основных принципов булевой алгебры является принцип двойственности, в соответствии с которым для каждого правила этой алгебры существует другое, двойственное ему, получающееся из исходного путем механической замены в нем операций AND на OR, OR на AND, а также констант T на F и наоборот. Приведенные ниже основные свойства булевой алгебры сгруппированы согласно этому принципу:

Свойство коммутативности

$$P \text{ AND } Q = Q \text{ AND } P, P \text{ OR } Q = Q \text{ OR } P$$

Перестановка высказываний, входящих в операции AND и OR, не влияет на результат

Свойство ассоциативности

$$P \text{ AND } (Q \text{ AND } R) = (P \text{ AND } Q) \text{ AND } R$$

$P \text{ OR } (Q \text{ OR } R) = (P \text{ OR } Q) \text{ OR } R$
Последовательность выполнения нескольких операций AND или нескольких OR на результат не влияет

Свойство дистрибутивности

$$\begin{aligned}P \text{ AND } (Q \text{ OR } R) &= (P \text{ AND } Q) \text{ OR } (P \text{ AND } R) \\ P \text{ OR } (Q \text{ AND } R) &= (P \text{ OR } Q) \text{ AND } (P \text{ OR } R)\end{aligned}$$

Операция AND дистрибутивна относительно операции OR и наоборот

Свойства операций AND и OR с константами

$$P \text{ AND } T = P, P \text{ AND } F = F, P \text{ OR } T = T, P \text{ OR } F = P$$

Свойства операции отрицания

$$P \text{ AND NOT } P = F, P \text{ OR NOT } P = T$$

4.3. ЛОГИЧЕСКОЕ СЛЕДОВАНИЕ (ИМПЛИКАЦИЯ)

Рассмотрим высказывания типа «Если P , то Q ». Частным случаем высказывания данного типа служит, например, следующее: «Если дождь идет, то трава растет». В приведенном примере P — «идет дождь», а Q — «трава растет». Следует отметить, что таблица истинности для этого высказывания должна включать в себя его значения для всех комбинаций значений составляющих. Т. е. истинность высказывания типа «Если P , то Q » зависит от того, как интерпретируются значения P и Q . Например, можно обнаружить, что значение всего высказывания — ложь, когда значение P равно истине, а Q — лжи, что для приведенного выше примера соответствует тому факту, что при дожде трава не растет.

Итак, мы имеем три логические величины: само высказывание и две его составляющие. Каждая из этих величин может произвольно принимать значение истина или ложь. Дождь может идти и не идти, трава может расти и не расти и, наконец, истина или ложь может заключаться в суждении, что если идет дождь, то непременно растет трава. В логике нет ничего, что могло бы нас убедить в истинности высказываний типа «Если P , то Q ». Проверить его истинность можно только на практике, исходя из сведений о реальном мире. Мы можем, например, обнаружить, что данное высказывание принимает значение ложь при условии, что P истинно, а Q ложно. Для приведенного примера это соответствует ложности суждения, что когда идет дождь, трава не вырастет. Можно убедиться, что для всех других комбинаций значений P и Q приведенное высказывание имеет значение истина.

Для начинающих изучение логики очень важно понять, что ее цель не столько проверка правильности произвольных суждений о реальном мире, сколько дедуктивный вывод значения одних высказываний из других. По высказыванию «Если P , то Q » в случае, когда P ложно, нельзя судить об истинности Q . В приведенном примере этот случай соответствует тому, что если дождь не идет, трава может расти и не расти — и то, и другое может быть истиной. Поэтому для рассмотренного случая, т. е. при условии, что P принимает значение ложь и при любом значении Q ,

Таблица 4.4

Истинность операций IMPLIES (импликация) и IFF (тождества)

P	Q	$P \text{ IMPLIES } Q$	$P \text{ IFF } Q$
T	T	T	T
T	F	F	F
F	T	T	F
F	F	T	T

все высказывание остается истинным. Однако, если данное высказывание истинно, то из истинности P необходимо следует истинность Q .

Операция, соответствующая высказываниям типа «Если P , то Q », называется логическим следованием, или импликацией, и обозначается $P \text{ IMPLIES } Q$.

В логике есть другая операция, напоминающая импликацию и задаваемая высказываниями « P тогда и только тогда, когда Q » или «Для P необходимо и достаточно Q ». Эта операция называется логическим тождеством и обозначается IFF .

Таблицы истинности для обеих рассмотренных операций представлены в табл. 4.4.

4.4. ЛОГИЧЕСКИЕ СИМВОЛЫ

До настоящего времени мы, насколько возможно, старались избегать использования символов при записи высказываний. Однако в классической логике, как и в булевой алгебре, существуют соответствующие правила символьной записи высказываний, позволяющие представлять логические высказывания в стандартной форме. В табл. 4.5 приведены наиболее часто используемые в булевой алгебре и в логике символьные обозначения логических операций.

В булевой алгебре операция отрицания обозначается горизонтальной чертой над переменной или выражением. Например, \bar{A} обозначает NOT A , а запись $\overline{A \cdot B}$ эквивалентна высказыванию NOT (A AND B). Последняя операция имеет самостоятельное название И-НЕ (или NAND).

Аналогично, операция $\overline{A + B}$, обозначающая высказывание НЕ (A ИЛИ B), обычно называется ИЛИ-НЕ (NOR). Таблицы истинности операций NAND и NOR приводятся в табл. 4.6.

Следует отметить, что таблицы истинности для NAND и NOR

Таблица 4.6
Операции NAND и NOR

A	B	$\overline{A \cdot B}$	$\overline{A + B}$
T	T	F	F
T	F	T	F
F	T	T	F
F	F	T	T

Таблица 4.5

Символы логики

Область	AND	OR	NOT	IMPLIES	IFF
Булевая алгебра	\cdot	$+$	$-$	Символа нет	Символа нет
Логика	\wedge	\vee	\sim	\rightarrow	\leftrightarrow

Примеры записи высказываний в булевой алгебре и логике

Функция	Булева алгебра	Логика
A AND B	$A \cdot B$	$A \wedge B$
A OR B	$A + B$	$A \vee B$
NOT A	\bar{A}	$\sim A$
NOT (A OR B)	$\overline{A + B}$	$\sim (A \vee B)$
NOT A OR B	$\bar{A} + B$	$\sim A \vee B$
NOT A AND B	$\bar{A} \cdot B$	$\sim A \wedge B$
NOT (A AND B)	$\overline{A \cdot B}$	$(A \wedge B)'$
A IMPLIES B		$A \leftrightarrow B$
A IFF B		$A \rightarrow B$

могут быть получены из таблиц истинности для операций AND и OR путем замены их значений в соответствующих строках на противоположные. В табл. 4.7 приведено несколько примеров записи логических операций.

5. ЛОГИЧЕСКИЕ ФУНКЦИИ ОТ ДВУХ ПЕРЕМЕННЫХ

Элементарные логические высказывания, которые не зависят друг от друга и могут произвольно принимать значения истина или ложь, будем называть логическими переменными. Высказывания, составленные с помощью логических операций над логическими переменными, будем называть логическими функциями.

Для двух логических переменных возможны четыре различные комбинации их значений: TT, TF, FT и FF. Множество комбинаций значений переменных какой-либо логической функции, при которых данная функция принимает значение истина, называется множеством истинности этой функции. Например, множество истинности для функции AND состоит из одного элемента (TT), тогда как для функции OR — из трех элементов (TT, TF, FT).

Упражнение 4.1

Самостоятельно запишите множества истинности для следующих функций: а) NAND; б) NOR; в) XOR; г) $A \cdot B$; д) $\sim A$ и е) $A \rightarrow B$.

Существует всего 16 различных способов заполнения таблиц истинности логических функций от двух переменных. В табл. 4.8 представлены все функции указанного вида от переменных x

Функции от двух переменных

p	q	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	r
T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
T	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
F	T	T	T	F	F	T	T	F	F	T	F	F	F	T	T	F	F
F	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F

и q. Следует отметить, что любая логическая функция от произвольного числа переменных может быть представлена в виде комбинации функций от двух переменных.

Напомним, что запись p означает: «высказывание p — истинно», а $\sim p$ — «высказывание p — ложно».

Покажем, как пишется символьная запись для функций, имеющих множества истинности, содержащие по одному элементу. Рассмотрим функцию h из табл. 4.8. Высказывание, соответствующее данной функции, истинно только тогда, когда p и q одновременно равны T , поэтому указанная функция может быть записана как $h = p \wedge q$ или в булевой форме: $h = p \cdot q$. Высказывание, соответствующее функции o , истинно, когда p и q одновременно ложны. Поэтому $o = \sim p \wedge \sim q$ или в булевой форме: $o = \bar{p} \cdot \bar{q}$. Заметим, что высказывание o не совпадает с $\sim(p \wedge q)$.

В качестве упражнения попробуйте, не заглядывая в книгу, дальше найти в табл. 4.8 функцию $\sim(p \wedge q)$. Попробуйте также записать в символьном виде какие-либо из функций, представленных в табл. 4.8.

Две оставшиеся нерассмотренными функции, имеющие по одному элементу в своих множествах истинности, обозначены в табл. 4.8 l и n . Функция l равна T только при одной комбинации значений входных переменных: TF . Это значит, что l истинна, когда p истинно, а q ложно, т. е. $l = p \wedge \sim q$. Аналогично можно установить, что $n = \sim p \wedge q$.

Теперь рассмотрим четыре функции от двух переменных, принимающих на одной комбинации значений p и q значение не T как в рассмотренных выше случаях, а F . Эти функции обозначены в табл. 4.8 через b , c , e и i . В данном случае удобнее сначала искать выражение для соответствующих противоположных функций, равных T при тех входных комбинациях, при которых исходные функции равны F , а затем с помощью отрицания этих функций получить искомую. Так, например, функция i равна F при $p = q = T$. Тогда $i = \sim(p \wedge q)$.

Упражнение 4.2

Самостоятельно найти функции b , c и e из табл. 4.8. Они представляют собой отрицания ранее рассмотренных функций. Как их назвать?

Из шестнадцати таблиц истинности логических функций от двух переменных восемь мы уже рассмотрели. Со следующими двумя, а и г, разобраться довольно просто. Функция а при всех значениях р и q истинна и представляет все тавтологии, например $p \vee \sim p$. Функция г всегда ложна и представляет все противоречия, например $p \wedge \sim p$.

Функции d и m также элементарны. Первая из них истинна, если истинно р, и ложна, если ложно р, т. е. $d = p$. Функция m наоборот: истинна, если ложно р, и ложна, если истинно р; следовательно, $m = \sim p$. По аналогии с функциями d и m легко заметить, что $f = q$, а $k = \sim q$.

Итак, в табл. 4.8 остались не рассмотренными две функции g и j. Множество истинности для j состоит из двух элементов — TF и FT. Данная функция может быть выражена в следующем символическом виде: $j = (p \wedge \sim q) \vee (\sim p \wedge q)$. Функция g может быть записана как отрицание функции j или иначе: $(p \wedge q) \vee (\sim p \wedge \sim q)$. Отметим, что j представляет собой рассмотренную ранее операцию XOR, тогда как g — операцию IFF.

Внимательно рассмотрев табл. 4.8, можно увидеть, что она обладает определенным рода симметрией относительно середины, т. е. линии, разделяющей столбцы h и i. Столбцы, находящиеся на одинаковом расстоянии от указанной линии (h и i, g и j и т. д.), получаются друг из друга с помощью операции отрицания. Поэтому для получения всех шестнадцати функций достаточно иметь выражения лишь для восьми левых (или правых) основных функций. Остальные восемь могут быть получены с помощью отрицания основных.

Упражнение 4.3

Получите символичные выражения для восьми левых (или правых) функций табл. 4.8, а затем, используя свойство симметрии, выразите оставшиеся функции.

Упражнение 4.4

Найдите в табл. 4.8 столбцы, соответствующие функциям $p \rightarrow q$ и $p \leftrightarrow q$.

4.6. ИСПОЛЬЗОВАНИЕ ПРОЛОГА ДЛЯ ДОКАЗАТЕЛЬСТВА ТЕОРЕМ

Для получения программы 4.1 вам необходимо загрузить модули Пролог-системы PROLOG, SIMPLE и TOLD. Функции AND, OR и NOT встроены в Пролог. С помощью этих функций и определяемых в программе отношений true, false и T мы сможем определять истинность сложных высказываний, а также проверять на эквивалентность различные высказывания.

В программе 4.1 отношения b , g и h соответствуют одноименным функциям, приведенным на табл. 4.8. Они могут быть представлены как $b = p \vee q$, $g = (p \wedge q) \vee (\sim p \wedge \sim q)$ и $h = p \wedge q$. Для того чтобы предоставить пользователю возможность вводить значение той или иной логической переменной, используется отношение `is-told`. Запрос может иметь следующую форму « p истинно?» (« p true?»), и, если на него будет получен ответ «да» (yes), то значением p становится истина (true). В случае же, если вводится ответ «нет» (no), значением p считается ложь (false).

Программа 4.1

```
X true if
    (X true) is-told
[X истина если в ответ на вопрос системы (X истинно?) пользо-
ватель введет «да» («yes»)]
X false if
    not X true
[X ложь если НЕ X истинна]
b T if
    (either p true or q true)
[b T, если ( p истинна ИЛИ q истинна)]
g T if
    (either p true and q true or p false and q false)
[g T, если ( p истинна И q истинна ИЛИ p ложь И q ложь)]
h T if
    p true and
    q true
[h T, если p истинна И q истинна]
```

Отношение `T`, как уже было указано, используется для проверки истинности сложных высказываний. С помощью данного отношения определяется истинность заданного сложного высказывания в зависимости от значений входящих в него переменных, которые в ответ на вопросы системы вводятся пользователем. Диалог между Пролог-системой и пользователем может иметь, например, следующий вид:

is (h T)	; Запрос пользователя
{ верно (h T)? }	
p true ? yes	; Вопрос Пролог-системы и ответ поль-
{ p истинно? да }	; зователя
q true ? yes	; Вопрос Пролог-системы и ответ поль-
{ q истинно? да }	; зователя
YES	; Ответ Пролог-системы
{ ДА }	
&.	

В рассмотренном примере диалога проверяется высказывание, составленное с помощью операций `AND`. Аналогичная проверка

для других значений переменных будет выглядеть следующим образом:

```
is (h T)
[верно (h T) ?]
p true ? yes
[r истинно ? да]
q true ? no
[q истинно ? нет]
NO
[HET]
& .
is (h T)
[верно (h T) ?]
p true ? no
[r истинно ? нет]
NO
[HET]
& .
```

Обратите внимание на то, что, обработав третий ответ пользователя, Пролог-система ответила сразу же после получения значения *p*, не производя в данном случае ненужной проверки значения *q*. Приведем теперь пример определения истинности высказывания *b*:

```
is (b T)
[верно (b T) ?]
p true ? yes
[r истинно ? да]
YES
[ДА]
& is (b T)
[верно (b T) ?]
p true ? no
[r истинно ? нет]
q true ? yes
[q истинно ? да]
YES
[ДА]
& is (b T)
[верно (b T) ?]
p true ? no
[r истинно ? нет]
q true ? no
[q истинно ? нет]
NO
[HET]
& .
```


Функция OR принимает значение истина, когда это же значение имеет хотя бы одна из двух ее переменных. Поэтому Пролог-система, обнаружив, что значением первой переменной является истина, прекращает проверку. Ниже приведены примеры диалога при проверке высказывания $g = ((p \wedge q) \vee (\sim p \wedge \sim q))$.

Случай 1

is (g T)
[верно (g T) ?]
p true ? yes
[p истинно ? да]
q true ? yes
[q истинно ? да]
YES
[ДА]

Случай 2

& is (g T)
[верно (g T) ?]
p true ? yes
[p истинно ? да]
q true ? no
[q истинно ? нет]
p true ? yes
[p истинно ? да]
NO
[НЕТ]

Случай 3

& is (g T)
[верно (g T) ?]
p true ? no
[p истинно ? нет]
p true ? no
[p истинно ? нет]
q true ? yes
[q истинно ? да]
NO
[НЕТ]

Случай 4

& is (g T)
[верно (g T) ?]
p true ? no
[p истинно ? нет]
p true ? no
[p истинно ? нет]

q true ? no
[q истинно ? нет]
YES
[ДА]
&.

В рассмотренном примере представлены варианты получения ответа для всех четырех позиций таблицы истинности данной функции. В первом варианте p и q истинны, поэтому второй терм анализируемой функции не проверяется, так как функция OR, связывающая оба терма, в случае истинности первого терма также принимает значение истина.

Значением первого терма во втором варианте является ложь, поэтому Пролог-система переходит к проверке истинности второго терма. Поскольку мы не обеспечили запоминание ответа на первый вопрос об истинности p ($p \text{ true ?}$), он повторяется еще раз, и мы должны еще раз утвердительно на него ответить. Сразу после этого Пролог-система устанавливает, что значением g является ложь, так как независимо от значения q значение второго терма — ложь.

В третьем и четвертом вариантах, так же как и во втором, Пролог-система повторяет вопрос « $p \text{ true ?}$ », чтобы определить значение второго терма функции g . Как будет показано в дальнейшем, можно избежать повторений вопросов, но сейчас мы не будем останавливаться на этом.

С помощью только что разобранных примеров было показано, как можно использовать Пролог для проверки правильности логических уравнений, описывающих логические функции b , g и h .

Можно воспользоваться средствами Пролога для проверки правильности альтернативных выражений для этих же функций. Для этого воспользуемся правилом алгебры логики, позволяющим манипулировать знаком отрицания: двойное отрицание переменной равно самой переменной: $\sim(\sim A) = A$. Пусть $\sim A = B$, тогда, взяв отрицание от обеих частей этого равенства и применив к левой части правило двойного отрицания, получим $A = \sim B$. Рассмотрим логическую функцию $b = p \vee q$. Согласно правилам алгебры логики, выражение для этой функции может быть преобразовано к следующему виду: $\sim b = \sim(p \vee q)$. Ранее уже было указано, что символьное выражение для функций можно искать с помощью отрицания функции, являющейся по отношению к исходной альтернативной, т. е. принимающей значение истина при тех значениях переменных, где исходная функция ложна, и наоборот. Функция OR принимает значение ложь только в том случае, когда p и q ложны одновременно, поэтому справедливо следующее равенство $\sim b = \sim p \wedge \sim q$. С помощью программы 4.1 проверим справедливость последнего равенства, а также сравним его с $\sim b = \sim(p \vee q)$. Если они окажутся идентичными,

это будет доказательством того, что $\sim p \wedge \sim q$ тождественно равно $\sim (p \vee q)$.

Добавим к программе 4.1 следующий фрагмент:

```
b F if
    p false and
    q false
[b F, если p ложь И q ложь]
```

Поскольку справедливо «X ложь, если НЕ X истина» (X false if not X true), приведенный выше фрагмент эквивалентен следующему:

```
b F if
    not p true and
    not q true
[b F, если НЕ p истина И НЕ q истина],
```

что в символьной записи соответствует $b = \sim p \wedge \sim q$. Проверим это равенство с помощью следующих вопросов:

```
is (b F)
[верно (b F) ?]
p true ? yes
[p истинно ? да]
NO
[HET]
& is (b F)
[верно (b F) ?]
p true ? no
[p истинно ? нет]
q true ? yes
[q истинно ? да]
NO
[HET]
& is (b F)
[верно (b F) ?]
p true ? no
[p истинно ? нет]
q true ? no
[q истинно ? нет]
YES
[ДА]
&.
```

Теперь удалим из программы отношение F с помощью команды kill F и введем другое его определение:

```
b F if
    not (either p true or q true)
[b F, если НЕ (p истина ИЛИ q истина)]
```

Ниже приведен диалог, служащий для проверки введенного отношения:

```
is (b F)
[верно (b F) ?]
p true ? yes
[r истинно ? да]
NO
[HET]
& is (b F)
[верно (b F) ?]
p true ? no
[r истинно ? нет]
q true ? yes
[q истинно ? да]
NO
[HET]
is (b F)
[верно (b F) ?]
p true ? no
[r истинно ? нет]
q true ? no
[q истинно ? нет]
YES
[ДА]
&.
```

О том, что данное отношение эквивалентно удаленному, свидетельствует совпадение последних двух диалогов.

4.7. ПОЛУЧЕНИЕ ТАБЛИЦ ИСТИННОСТИ С ПОМОЩЬЮ ПРОЛОГА

Более мощными возможностями обладает приведенная ниже программа 4.2, позволяющая генерировать таблицы истинности для функций от двух переменных. Эта программа может быть, например, использована для проверки тождественности двух альтернативных форм представления какой-либо функции. В программе используется бинарное отношение $X \text{ true } (YZ) [X \text{ истина } (YZ)]$ для проверки истинности функций.

Для демонстрации предлагаемого способа выбраны следующие функции: $A = X \cdot Y$, $B = X + Y$ и $C = X + \bar{Y}$ (для обозначения функций используются большие буквы, что позволяет отличать их от обозначений из табл. 4.8).

Программа 4.2

```
T true           ; T истина
A true if        ; A истина если
X true and       : X истина И
```

Y true	; Y истина
B true if	; B истина если
(either X true or Y true);	(X истина ИЛИ Y истина)
C true if	; C истина если
(either X true or Y false);	(X истина ИЛИ Y ложь)
F false	; F ложь

Ниже приведен пример использования данной программы:

```

&all (x y : A true (x y))
[Определить все (x y : A истина (x y))]
T T
No (more) answers
[Ответов (больше) нет]
&all (x y : B true (x y))
[Определить все (x y : B истина (x y))]
T X
X T
No (more) answers
[Ответов (больше) нет]
&all (x y : C true (x y))
[Определить все (x y : C истина (x y))]
T X
X F
No (more) answers
[Ответов (больше) нет]

```

Пролог-система выдает значения x и y , соответствующие только истинным значениям каждой заданной функции. Все остальные значения не рассматриваются. Для функций B и C получены ответы (TX, XT) и (TX, XF) соответственно. Символ X в ответе означает, что на его месте может находиться как F , так и T . Таким образом, указанные две последовательности ответов означают, что функции B и C принимают значение истина на множествах значений аргументов (TT, TF, FT) и (TT, TF, FF) соответственно.

С помощью следующих дополнений в программу 4.2 можно получить множества значений аргументов, для которых функции A , B и C принимают значение ложь.

X false (Y Z) if	; X ложь (Y Z) если
Y val and	; Y значимо И
Z val and	; Z значимо И
not (X true (Y Z))	; НЕ (X истина (Y Z))
X val if	; X значимо, если
(either X true or X	; (X истина ИЛИ X ложь)
false);	

На следующие три вопроса:

- 1) all (x y : A false (x y))
[Определить все (x y : A ложь (x y))]
- 2) all (x y : B false (x y))
[Определить все (x y : B ложь (x y))]
- 3) all (x y : C false (x y))
[Определить все (x y : C ложь (x y))]

будут получены такие ответы:

- | | | |
|--------|--------|--------|
| 1) T F | 2) F F | 3) F T |
| F T | | |
| F F | | |

Заметим, что отношение val (значимо) служит для генерации всех возможных комбинаций значений входных переменных для последующей их проверки с помощью отношения false. Без этого отношения Пролог-система не смогла бы выдать ни одного ответа в тех случаях, когда значением заданных функций является ложь, и отвечала бы пользователю «No more answers [«Ответов больше нет»]. Читатель может убедиться в этом сам, удалив отношения val из программы и повторив запросы.

Можно получить таблицу истинности, содержащую не только значения двух переменных, но и соответствующие им значения функций. Определим новое отношение tab (таблица), построенное с помощью отношений true и false. Для этого расширим предыдущую программу и в результате получим программу 4.3:

Программа 4.3

T true	; T истина
A true (X Y) fi	; A истина (X Y), если
X true and	; X истина И
Y true	; Y истина
B true (X Y) if	; B истина (X Y), если
(either X true or Y	; (X истина ИЛИ Y ложь)
true)	
C true (X Y) if	; C истина (X Y), если
X true (Y Z x) if	; X истина (Y Z x), если
X true (Y Z) and	; X истина (Y Z) И
x true	; x истина
X val if	; X значимо, если
(either X true or X	; (X истина ИЛИ X ложь)
false)	
X tab (Y Z x) if	; X таблица (Y Z x), если
(either X true (Y Z x) or X false (Y Z x))	
; (X истина (Y Z x) ИЛИ X ложь (Y Z x))	
F false	; F ложь
X false (Y Z) if	; X ложь (Y Z), если
Y val and	; Y значимо И
Z val and	; Z значимо И

not (X true (Y Z))	; HE (X (Y Z))
X false (Y Z x) if	; X ложь (Y Z x), если
X false (Y Z) and	; X ложь (Y Z) И
x false	; x ложь

Ниже приведены ответы Пролог-системы на запросы, позволяющие получить таблицы истинности для функций А, В и С, включающие в себя третий столбец из значений функций:

1) all (x y z : A tab (x y z))

[Определить все (x y z : A таблица (x y z))]

2) all (x y z : B tab (x y z))

[Определить все (x y z : B таблица (x y z))]

3) all (x y z : C tab (x y z))

[Определить все (x y z : C таблица (x y z))]

1) T T T	2) T X T	3) T X T
T F T	X T T	X F T
F T F	F F F	F T F
F F F		

Можно убедиться, что полученные таблицы соответствуют функциям А, В и С. Символ Х в ответах заменяет одновременно Т и F. Таким образом, строка с одним таким символом соответствует двум обычным строкам. Первые два столбца содержат значения переменных, а последний — значения функций.

Выше было показано, как можно с помощью Пролога исследовать некоторые не всегда очевидные свойства логических функций. В табл. 4.9 приведены уже рассмотренные выше правила алгебры логики.

Таблица 4.9

Коммутативность	$p \cdot q = q \cdot p$
Ассоциативность	$p + q = q + p$
	$(p \cdot q) \cdot r = p \cdot (q \cdot r)$
Дистрибутивность	$(p + q) + r = p + (q + r)$
	$p \cdot (q + r) = p \cdot q + p \cdot r$
Двойное отрицание	$p + (q \cdot r) = (p + q) \cdot (p + r)$
Идемпотентность	$p = p$
	$p \cdot p = p$
Поглощение	$p + p = p$
Несколько дополнительных аксиом	$p \cdot (p + q) = p$
	$p \cdot T = p$
	$p \cdot F = F$
	$p + T = T$
	$p + F = p$
	$p \cdot p = F$
	$p + \bar{p} = T$

Упражнение 4.5

Запишите правила логики из табл. 4.9 с помощью операций AND, OR и NOT.

Напомним, что в процессе выполнения логических операций необходимо учитывать их приоритеты. В первую очередь выполняются операции в скобках, затем, в порядке убывания приоритетов, следуют NOT, AND, OR и (в логике) IMPLIES на одном уровне с IFF. С помощью скобок можно изменять порядок вычисления логических выражений. Поэтому, например, выражение $p + q \cdot r$ не равно $(p + q) \cdot r$.

4.8. ПРАВИЛА ДЕ МОРГАНА

В задачах, решаемых с помощью алгебры логики, часто используются два правила, известных под названием правил де Моргана. Приведем формулировку этих правил:

$$\overline{P \cdot Q \cdot R \dots} = \bar{P} + \bar{Q} + \bar{R} + \dots$$

$$\overline{P + Q + R + \dots} = \bar{P} \cdot \bar{Q} \cdot \bar{R} \dots$$

Многоточия, использованные в приведенных формулах, означают, что в выражении может содержаться произвольное число переменных. Вот несколько примеров использования данных правил:

$$1. \overline{\bar{A} + \bar{B}} = A \cdot B$$

$$2. \overline{\bar{A} + B \cdot C} = A \cdot \overline{B \cdot C} = A \cdot (\bar{B} + \bar{C})$$

$$3. A + C + D = A + \overline{\bar{C} \cdot \bar{D}} = \overline{\bar{A} \cdot \bar{C} \cdot \bar{D}}$$

Первый пример достаточно прост. Во втором примере сначала выражение $\overline{\bar{A} + X}$, где $X = B \cdot C$, преобразуется к виду $\bar{A} \cdot \bar{X}$, а затем $\bar{X} = \overline{B \cdot C}$ приводится к виду $\bar{B} + \bar{C}$. В третьем примере с помощью первого правила из $C + D$ получаем $\overline{\bar{C} \cdot \bar{D}}$, а затем выводим окончательный ответ.

Применение правил де Моргана обычно сводится к последовательному выполнению следующих трех шагов:

- а) отрицание всех отдельных термов выражения;
- б) замена операций AND на OR и наоборот;
- в) отрицание всего выражения.

На практике правила де Моргана используются для представления произвольных логических функций в виде, удобном для реализации их с помощью электронных логических элементов одного типа.

В частности, все функции из табл. 4.8 можно представить с помощью суперпозиции функций И-НЕ (NAND) или ИЛИ-НЕ

(NOR). Например, функция $h = p \cdot q$ (AND) может быть выражена с помощью NAND в виде $h = \overline{p \cdot q}$ или с помощью NOR в виде $h = \overline{p + q}$. Функция отрицания реализуется в данном случае с помощью свойства поглощения, например, следующим образом: $\overline{p} = \overline{p + p}$. Эти преобразования применяются на практике при проектировании аппаратуры вычислительных устройств, что позволяет рационально использовать элементную базу. В заключение добавим, что любые логические функции могут быть представлены также с помощью функций IMPLIES и NOT.

Упражнение 4.6

Запишите следующие выражения в двух вариантах с помощью NAND (а) и NOR (б):

- 1) $A \cdot \overline{B}$, 2) $\overline{A} + B$, 3) $A \cdot (B + C)$,
4) $A + \overline{B \cdot C}$, 5) $A \cdot B + C$, 6) $\overline{A} + \overline{B}$.

4.9. ИСПОЛЬЗОВАНИЕ ПРОЛОГА ДЛЯ РЕАЛИЗАЦИИ ПРАВИЛ ДЕ МОРГАНА

В программе 4.4 функции a , b , c , d , q и h представлены в трех различных вариантах, иллюстрирующих применение правил де Моргана. Различным вариантам соответствуют отношения, обозначенные разным числом букв. Например, отношение a — первый вариант функции a из табл. 4.8, aa — второй вариант той же функции и aaa — третий. Теперь, если три варианта представляют одну и ту же функцию, реакция Пролог-системы в случае одних и тех же переменных будет одинаковой.

Программа 4.4

```
a T if
    (either p true or p false)
aa T if
    not (p true and p false)
aaa T if
    not not (either p true or p false)
b T if
    (either p true or q true)
bb T if
    not (p false and q false)
bbb T if
    not not (either p true or q true)
c T if
    (either p true or q false)
cc T if
    not (p false and q true)
```

```

ccc T if
    not not (either p true or q false)
d T if
    p true
dd T if
    not (p false and p false)
ddd T if
    not not (either p true or p true)
g T if
    (either p true or q true or p false and q false))
gg T if
    not not (p true or q true) and not (p false and q false)
ggg T if
    not not (either not (either p false or q false) or not (either
    p true or q true))
h T if
    p true and
    q true
hh T if
    not not (p true and q true)
hhh T if
    not (either p false or q false)
X true if
    (X true) is-told
X false if
    not X true
X F if
    not X T

```

В данной программе снова используется модуль TOLD, позволяющий организовать диалог пользоваться с Пролог-системой. Ниже приводится пример такого диалога:

```

is (h T)
[верно (h T)]
p true ? yes
[p истинно? да]
q true ? yes
[q истинно? да]
YES
[ДА]
& is (hh T)
[верно (hh T)]
p true ? yes
[p истинно? да]
q true ? yes
[q истинно? да]
YES
[ДА]

```

```

& is (hhh T)
[верно (hhh T)]
o true ? yes
[r истинно? да]
q true ? yes
[q истинно? да]
YES
[ДА]

```

Указанный диалог показывает, что первая строка таблицы истинности для высказываний h , hh и hhh одна и та же. Аналогичным образом могут быть проверены и остальные строки и таблицы.

Ниже приводится программа 4.5, полученная в результате модификации программы 4.3. Эта программа позволяет проверять таблицы истинности для различных вариантов представления функций алгебры логики.

Программа 4.5

```

T true
X true (Y Z x) if
    X true (Y Z) and
    x true
b true (X Y) if
    (either X true or Y true)
bb true (X Y) if
    X val and
    Y val and
    not (X false and Y false)
bbb true (X Y) if
    X val and
    Y val and
    not not (either X true or Y true)
c true (X Y) if
    (either X true or Y false)
cc true (X Y) if
    X val and
    Y val and
    not (X false and Y true)
ccc true (X, Y) if
    X val and
    Y val and
    not not (either X true or Y false)
h true (X Y) if
    X true and
    Y true
hh true (X Y) if
    X val and

```

```

    Y val and
    not not (X true and Y true)
hhh true (X Y) if
    X val and
    Y val and
    not (either X false or Y false)
X val if
    (either X true or X false)
X tab (Y Z x) if
    (either X true (Y Z x) or X false (Y Z x))
F false
X false (Y Z) if
    Y val and
    Z val and
    not (X true (Y Z))
X false (Y Z x) if
    X false (Y Z) and
    x false

```

Для получения с помощью программы 4.5 таблицы истинности для функции *b* необходимо ввести запрос типа *all (xyz : b tab (xyz))*. Затем после ввода аналогичных запросов для функций *bb* и *bbb* получаем соответствующие им таблицы.

Ниже приводится диалог, возникающий после ввода трех указанных запросов.

```

& all (x y z : b tab (xyz))
TXT
XTT
FFF
& all (x y z : bb tab (x y z))
TTT
TFT
FTT
FFF
& all (x y z : bbb tab (x y z))
TTT
TFT
FTT
FFF
No (more) answers

```

По полученным таблицам можно убедиться в идентичности функций *b*, *bb* и *bbb*.

4.10. ЛОГИЧЕСКОЕ СЛЕДОВАНИЕ В ПРОЛОГЕ

Истинность логического высказывания может быть определена после того, как на основании каких-либо общеизвестных фактов установлена истинность аргументов, входящих в данное выска-

зывание. Например, известно, что солнце всходит на востоке. Из этого можно вывести, что если одна из двух точек земной поверхности находится восточнее другой, то из нее восход солнца можно наблюдать раньше. Однако логически строгого доказательства того, что солнце всходит на востоке, может не быть.

Применение методов логического вывода позволяет получать новые результаты на основе уже известных фактов и теорий. Приведем несколько простых примеров.

1. Джим человек. Все люди имеют мозг. Следовательно, Джим имеет мозг.
2. Джим человек. Все люди имеют по 5 ног. Следовательно, у Джима 5 ног.
3. Джим человек. У Джима есть мозг. Следовательно, все люди имеют мозг.
4. Все люди смертны. Джим человек. Следовательно, Джим смертен.
5. Все люди смертны. Джим смертен. Следовательно, Джим человек.

Первая и вторая цепочки умозаключений являются корректными, хотя во второй из-за того, что одно из исходных утверждений ошибочно, в результате получилось абсурдное утверждение. Исходные утверждения третьей цепочки правильны, корректным получился и результат, но законы логики были нарушены: нельзя строить дедуктивный вывод от частного к общему. Четвертая цепочка представляет собой классический образец корректного логического вывода, а в пятой содержится ошибка: из того, что Джим смертен, не следует, что Джим человек.

Приведем еще два правила, которые помогут избежать рассмотренных ошибок.

1. *Правило «modus ponens»*. Согласно этому правилу из истинности высказывания «из Р следует Q» ($P \text{ IMPLIES } Q$) и из истинности Р выводится истинность Q.

Первый, второй и четвертый из приведенных примеров иллюстрирует применение этого правила. Данное правило более наглядно можно представить так:

Р
из Р следует Q
следовательно, Q

2. *Правило цепочки*. Если истинны «из Р следует Q» и «из Q следует R», то можно заключить, что «из Р следует R».

Например, из высказываний «все собаки — животные» и «животные не умеют думать» следует «собаки не умеют думать». Запишем это правило в следующем виде:

из Р следует Q
из Q следует R

следовательно,

из Р следует R

Программа 4.6 иллюстрирует применение только что описанных правил:

Программа 4.6

Alan man	[Алан мужчина]
Anna woman	[Анна женщина]
x human if	[x человек, если]
(either x man or x woman)	
	[x мужчина или x женщина]
x mortal if	[x смертен, если]
x human	[x человек]

В ответ на вопрос
is (Alan human)

[верно (Алан человек)]

Пролог-система ответит утвердительно, а на запрос

all (x : x human)

[определить все (x : x человек)]

будут получены ответы «Alan» и «Ann». Если ввести запрос

all (x : x mortal)

[определить все (x : x смертен)]

Пролог-система также выдаст ответы «Alan» и «Ann», полученные с помощью правила цепочки и «modus ponens».

Работа программы 4.6 иллюстрирует возможности Пролога делать «самостоятельные» логические выводы из заданных высказываний. Причем следует отметить, что точный алгоритм вывода не задается. В программе 4.7, построенной с помощью только двух отношений — true и implies, — правило modus ponens задается в явном виде, а применение правила цепочки реализуется встроенными средствами Пролога. Отметим, что правило цепочки можно расширить, например, следующим образом: если из P следует Q, из Q следует R, из R следует S, а из S следует T, то из любого из высказываний P, Q, R и S следует T.

Программа 4.7

(Alan is a man) true	[(Алан — мужчина) истина]
(Ann is a woman) true	[(Анна — женщина) истина]
X true if	[x истина, если]
Y implies X and	[из Y следует X и]
Y true	[Y истина]
(X is a man) implies (X is human)	
[из (X — мужчина) следует (X — человек)]	
(X is a woman) implies (X is human)	
[из (X — женщина) следует (X — человек)]	
(X is human) implies (X thinks)	
[из (X — человек) следует (X — мыслит)]	
(X thinks) implies (X is intelligent)	
[из (X — мыслит) следует (X — разумен)]	

(X is intelligent) implies (X can learn)
 [из (X — разумен) следует (X может обучаться)]
 (X is intelligent) implies (X can reason)
 [из (X — разумен) следует (X может рассуждать)]
 В ответ на запрос
 all (x : x true)
 [определить все (x : x истинно)]

программа 4.7 выведет из высказываний «Alan is a man» и «Ann is a woman», что Алан и Анна — люди. Отметим, что программирование правила modus ponens в другом виде:

X true if Y true and Y implies X
 [X истинно, если Y истинно и из Y следует X]

приведет к бесконечным возвратам при попытках вывести новые факты из уже выведенных и через некоторое время система сообщит о нехватке места в памяти. Читатель может в качестве упражнения проверить на практике, что получится при внесении указанных изменений в программу 4.7. Попробуйте также при внесении изменений в данное правило программы 4.7, кроме того, удалить из него условие Y true. Введите после этого запрос all (x : x true). Тогда кроме двух запрограммированных в явном виде фактов (первые две строки программы) будут выданы ответы типа:

X is human
 X thinks

и т. д. На запрос «is (David is human) true» будет получен утвердительный ответ. Другими словами, если логический вывод строится на основе аргументов, значения которых не определены, то в этом случае могут быть получены как истинные, так и ложные результаты, и логическая система становится бесполезной.

Программы 4.6 и 4.7 позволяют на достаточно простом уровне продемонстрировать некоторые свойства реляционных баз данных, которые более подробно будут рассматриваться позднее. В данной же главе мы обращаем внимание лишь на логические механизмы, заложенные в Пролог.

Говоря о программах 4.6 и 4.7, следует обратить внимание на следующие четыре момента.

1. Для достаточно большого числа отношений программа 4.6 может оказаться слишком длинной и не уместиться в памяти ЭВМ.

2. В программе 4.7 использовать отношение implies на самом деле не обязательно. Данное отношение применяется лишь для демонстрации правила цепочки и можно его заменить на условное предложение Пролога if. Однако в ряде случаев следует применять implies вместо if.

3. Всегда следует стремиться к общности для того, чтобы создавать более компактные и более эффективные программы.

Например, одно условие «if x is a y» может заменить несколько таких условий, как «if x is a man» и «if x is a woman» и т. д.

4. Списковые конструкции позволяют в компактной форме представлять сложные отношения.

При написании программы 4.8 были использованы более эффективные средства программирования, чем в рассмотренных выше двух программах. Заметим, что программа 4.8 строится с помощью отношения true и списковых структур.

Программа 4.8

(X is a man) true if	[(X мужчина) истина, если]
X ON (Alf Bert Colin)	[X из-списка (Альфред Берт Колин)]
(X is a woman) true if	[(X женщина) истина, если]
X ON (Anne Betty Clare)	[X из-списка (Анна Бетти Клер)]
(X is a child) true if	[(ребенок) истина, если]
X ON (David Helen Jim)	[X из-списка (Дэвид Элен Джим)]
(X is human) true if	[(X человек) истина, если]
Y ON (man woman child) and	[Y из-списка (мужчина женщина ребенок) и]
(X is a Y) true	[(X являются Y) истина]
(X is a dog) true if	[(X собака) истина, если]
X ON (Fido Rover Spot)	[X из-списка (Фидо Ровер Спот)]
(X thinks) true if	[(X мыслит) истина, если]
(X is human) true	[(X человек) истина]
(X can reason) true if	[(X может рассуждать) истина, если]
(X thinks) true	[(X мыслит) истина]
(X can learn Y) true if	[(X может научиться Y) истина, если]
(X can reason) true	[(X может рассуждать) истина]
(X can learn tricks) true if	[(X может научиться проделкам) истина, если]
(X is a dog) true	[(X собака) истина]

Данная программа может быть использована, например, для проверки какого-либо факта:

```
is ((Alf is a man) true)
[верно ((Альфред мужчина) истина)]
YES
[ДА]
```

В другом случае ее можно применять для поиска нужного элементарного даного:

```
one (x: (x is a child) true)
[определить один (x: (x ребенок) истина)]
```


David
[Дэвид]

Может быть введен запрос в более общей форме:

is ((x is a man) true)
[верно ((мужчина) истина)]
YES
[ДА]

В последнем случае программа подтверждает, что существует x , удовлетворяющее отношению «(x is a man) true». В другом случае на запрос

is ((x is a cat) true)
[верно ((х кошка) истина)]

будет получен отрицательный ответ «NO», свидетельствующий о том, что программа не может найти x , удовлетворяющее данному запросу.

На примере рассмотренной программы можно убедиться в способности Пролога делать логические выводы. Например:

is ((Hellen can reason) true)
[верно ((Элен может рассуждать) истина)]
YES
[ДА]

В данном случае Пролог-система «рассуждает» следующим образом:

все дети — люди,
все люди мыслят,
из «х — мыслит» следует «х может рассуждать»;
Элен — дитя,
следовательно, Элен может рассуждать.

Программа может также сделать следующий вывод:

is ((Alf can learn programming) true)
[верно ((Альфред может обучиться программированию) истина)]
YES
[ДА]

С помощью правила

(X can learn Y) true if (X can reason) true
[(X может научиться Y) истина, если (X может рассуждать) истина]

Пролог-система может сделать произвольное число логических выводов типа, рассмотренных выше. Вот еще несколько таких примеров:

is ((Fido can learn programming) true)
[верно ((Фидо может обучиться программированию) истина)]

NO

[HET]

all (X: (Fido can learn X) true

[определить все (X : (Фидо может обучиться X) истина)]

tricks

[проделки]

all (x: (x can learn programming) true)

[определить все (x: (x может обучиться программированию истина)]

Alf

Bert

Colin

и т. д.

Данная программа с точки зрения практического ее применения довольно далека от совершенства, поскольку ее работа основана на мнении, что человек может обучиться чему угодно. Более полезна будет программа, в которой более точно определена способность к обучению в более сложной форме. Для этого в программе 4.8 вместо восьмого предложения добавим следующий фрагмент:

(X is 1) true if [(X 1) истина если]

X ON (Alf Anne David)

[из-списка (Альфред Анна Дэвид)]

(X is 2) true if [(X 2) истина, если]

X ON (Bert Betty Helen)

[X из-списка (Берт Бетти Элен)]

(X is 3) true if [(X 3) истина, если]

X ON (Colin Clare Jim)

[X из-списка (Колин Клер Джим)]

(X is 1 subject) true if

X ON (walking talking digging)

[(X из-группы 1) истина, если]

[X из-списка (ходить говорить копать)]

(X is 2 subject) true if

X ON (reading writing arithmetic prolog)

[(X из-группы 2) истина если]

[X из-списка (читать писать арифметика пролог)]

(X is 3 subject) true if

X ON (physics algebra)

[(X из-группы 3) истина, если]

[X из-группы (физика алгебра)]

(Einstein is a genius) true

[(Эйнштейн гений) истина]

(X can learn advanced Y) true if

(X is a genius) true and

(Z can learn Y) true

[(X может научиться сложным Y) истина, если]

[(X гений) истина и (Z может научиться Y) истина]

```
(X can learn Y) true if
  (X is Z) true and
  (Y is Z subject) true
[(X может научиться Y) истина, если]
  [(X Z) истина и (Y из-группы Z) истина]
```

Теперь можно проверить работу данной программы с помощью следующего диалога:

```
is ((Betty can learn Prolog) true) true)
[верно ((Бетти может обучиться Прологу) истина)]
YES
[ДА]
is ((Betty can learn algebra) true)
[верно ((Бетти может обучиться алгебре) истина)]
NO
[НЕТ]
all (x: (Colin can learn x) true)
[определить все (x: (Колин может обучиться x) истина)]
walking
[ходить]
talking
[говорить]
и т. д.
is (Einstein can learn advanced tricks) true
[верно (Эйнштейн может обучиться сложным проделкам)
истина]
YES
[ДА]
```

Вообще изучать на практике логические механизмы Пролога очень полезно для начинающих. Это занятие довольно увлекательно и часто приводит к неожиданным результатам. Кроме того, такие упражнения способствуют лучшему пониманию логики языка. В качестве такого упражнения можно рекомендовать читателю проанализировать ответы на следующие вопросы:

```
all (x: x true)
[определить все (x: x истина)]
all (x y: (x y) true)
[определить все (x: (x y) истина)]
all (x y z: (x y z) true)
[определить все (x y z: (x y z) истина)]
all (x y: (x can y) true)
[определить все (x y: (x может y) истина)]
all (x y: (x can y) true)
[определить все (x y: (x может | y) истина)]
```

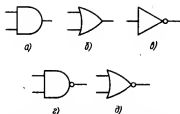


Рис. 4.1. Символические обозначения логических элементов:
 а — AND; б — OR; в — NOT; г — NAND; д — NOR

4.11. ПРОЛОГ И ЦИФРОВЫЕ ЛОГИЧЕСКИЕ УСТРОЙСТВА

Для того чтобы разобраться в описываемом ниже применении Пролога для построения электронных схем, реализующих логические функции, не нужно быть специалистом в электронике. Достаточно знать, что логические схемы представляют собой разновидность электронных схем, имеющих один или несколько вхо-

дов и один выход. Сигналы на выходе схемы зависят от различных комбинаций электрических потенциалов двух уровней, подаваемых на ее вход. Логические уровни, которые раньше задавались с помощью значений истина и ложь, определяются в логических схемах как уровни электрических потенциалов на входах и выходах этих схем.

Схемы, в которых более высокий уровень напряжения соответствует логической единице (истине), а низкий — логическому нулю (лжи), называют положительными логическими схемами. На выходе элементарной логической схемы (вентилia) устанавливается сигнал, соответствующий значению истина или ложь в зависимости от комбинации логических сигналов на входе. Из элементарных схем строятся более сложные. Примеры таких элементарных схем, реализующих функции AND, OR, NOT, NAND и NOR, представлены на рис. 4.1.

Все эти вентили, кроме одноходового инвертора (схема NOT), имеют по два входа. В программе 4.9 используется отношение `fn`, моделирующее работу схем NOT и AND. Преимущество предлагаемого метода заключается в простоте представления сложных схем. Чтобы обозначения вентиляей отличались от имен встроенных отношений Пролога, в программе для отношений, моделирующих элементы схем, используются имена, состоящие из первой большой буквы и остальных малых, например `And`, `Not` и т. д.

Программа 4.9

```
(Not 0) fn 1
(Not 1) fn 0
(And 0 | X) fn 0
(And 1) fn 1
(And 1 | X) fn Y if
(And | X) fn Y
```

Ниже приводится пример работы этой программы:

```
all (xyz: (And xy) fn z)
0 X 0
1 0 0
```

1 1 1

No (more) answers

Работа трехвходового элемента And моделируется с помощью запроса

all (xyz X: (And xyz) in X).

Результат всегда помещается в последнем столбце.

Упражнение 4.7

Написать программу, моделирующую работу вентиля Or, а затем с помощью отношения Not, а также отношений And и Or определить отношения Nand и Nor.

Приведенная ниже программа 4.10, построенная с помощью отношений tab и b, позволяет выводить на экран таблицы, описывающие зависимости выходных сигналов от входных для произвольных логических схем. Отношение tab выводит списки уровней входных и выходных сигналов, тогда как отношение b обеспечивает генерацию всех двоичных последовательностей, имеющих длину, соответствующую числу входных сигналов.

Программа 4.10

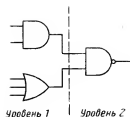
```
(X) b 1 if
      X ON (0 1)
X b Y if
      1 LESS Y and
      SUM(1 Z Y) and
      x b 1 and
      y b Z and
      APPEND (x y X)
tab (X Y Z x) if
      Z b X and
      fn ((Y|Z) x)
```

В отношении tab переменная Y обозначает название схемы, X — число входов, Z — комбинацию нулей и единиц, обозначающих уровни входных сигналов, а x — уровень выходного сигнала. Например, для получения таблицы зависимости выходного сигнала от входов схемы Nand (с помощью отношения из ответа к упражнению 4.7) необходим следующий запрос:

```
all (x y): tab (3 Nand x y)
(0 0 0) 1
(0 0 1) 1
... и т. д.
(1 1 1) 0
```

Теперь определим отношение, позволяющее анализировать работу сложной логической схемы. На рис. 4.2 представлена довольно простая логическая схема, имеющая два уровня, иа

Рис. 4.2. Двухуровневая логическая схема



первом из которых расположены два входных элемента: первый — And с двумя входами, другой — Or с тремя. На втором, выходном уровне размещен один элемент Nand. Ниже приводится определение отношения *com*, позволяющего описывать работу таких схем:

```
(X Y Z x y) com z if
  tab (X Z X1 Y1) and
  tab (Y x Z1 x1) and
  APPEND (x1 z1 y1) and
  (y Y1 X1) fn z1 and
  APPEND (y1 (z1) z)
```

В приведенном выше фрагменте программы таблицы уровней входных и выходных сигналов получаются с помощью отношения *tab*. Затем совокупность сигналов первого уровня используется для моделирования работы второго уровня схемы с помощью отношения *fn* и списка значений входных сигналов *z*. Окончательный ответ обеспечивается оператором Пролога *APPEND*. Так, для схемы, показанной на рис. 4.2, таблица может быть получена следующим образом:

```
all (X: (2 3 And Or Nand) com x)
(0 0 0 0 0 1)
(0 0 0 0 1 1)
... и т. д.
(1 1 1 1 0 0)
(1 1 1 1 1 0)
No (more) answers
```

Отметим, что первые две переменные используются для обозначения числа входов двух элементов первого уровня. Элемент второго уровня был уже описан ранее и имеет два входа.

Упражнение 4.8

Напишите запросы, позволяющие получать таблицы истинности логических схем, имеющих структуру:

а) два двухвходовых элемента Nand, соединенные на выходе с Nor;

б) два двухвходовых элемента And, соединенные на выходе с таким же And;

в) два инвертора (Not), соединенные с элементом Nand.

Упражнение 4.9

- а. Сравните результаты выполнения упражнений 4.8, а и б.
б. С помощью отношения `fp` получите таблицу истинности для двухвходового элемента `Or` и сравните результат с ответом в упражнении 4.8, в.

4.12. ИСПОЛЬЗОВАНИЕ ЛОГИЧЕСКИХ ЭЛЕМЕНТОВ ДЛЯ ДОКАЗАТЕЛЬСТВА ТЕОРЕМ

Упражнения 4.8 и 4.9 должны были подготовить читателя к изучению материала данного раздела. Отношения, описанные выше, могут быть использованы для доказательства теорем логики и булевой алгебры. Например, рассмотрим два частных случая применения одного из правил де Моргана:

$$\overline{A \cdot B} = \bar{A} + \bar{B}; \text{ (теорема 1)}$$

$$\overline{A + B} = \bar{A} \cdot \bar{B} \text{ (теорема 2)}$$

Для проверки теоремы 1 воспользуемся следующими двумя запросами:

```
all (x y : tab (2 Nand x y))
all (x: (1 1 Not Not Or) com x)
```

Сравнив приведенные ниже ответы Пролог-системы, убедимся в том, что анализируемые функции из левой и правой частей формулировки теоремы 1 идентичны:

(0 0) 1	(0 0 1)
(0 1) 1	(0 1 1)
(1 0) 1	(1 0 1)
(1 1) 0	(1 1 0)

Аналогичным способом можно доказать правильность теоремы 2. Рассмотренная программа может быть усовершенствована с помощью дополнительного отношения, которое позволяет проверять эквивалентность двух логических выражений. Это отношение называется `equiv` и имеет следующий вид:

```
(X Y | Z) equiv (X Y | x) if
  (forall ((X Y | x) com y then (X Y | Z) com y)
```

Эта запись означает, что две логические схемы эквивалентны, если они формируют одинаковые выходные сигналы для каждой

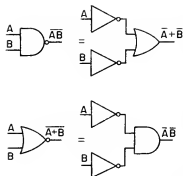


Рис. 4.3. Иллюстрация к теоремам 1 и 2

комбинации входных сигналов. В том случае, когда требуется проверить эквивалентность некоторой сложной схемы и элементарной, состоящей из одного элемента, удобно пользоваться фиктивным элементом — проводником Link, непосредственно соединяющим вход первого уровня со входом второго. Для этого дополним отношение `fn` двумя следующими строками:

(Link 0) `fn 0`
 (Link 1) `fn 1`

Программа Link соответствует элементу, повторяющему на выходе свой же входной сигнал. Теперь можно воспользоваться отношением `equiv`, например, следующим образом:

```
is ((1 1 Link Link Nand) equiv (1 1 Not Not Or)
YES
&.
```

Утвердительный ответ подтверждает правильность формулировки теоремы 1. Очевидно, что такой способ доказательства эквивалентности логических схем удобнее, чем рассмотренный раньше, при котором приходилось вручную сравнивать таблицы истинности. На рис. 4.3 показаны логические схемы, иллюстрирующие теоремы 1 и 2.

Упражнение 4.10

а. Проверьте теорему 2 с помощью отношения `equiv`.

б. С помощью отношения `tab` или `com` получить таблицы истинности левой и правой частей теоремы 2 и убедиться, что они совпадают.

Мы использовали программу 4.10 для того, чтобы подтвердить или отвергнуть какую-либо гипотезу о равенстве одних логических схем другим. Следующим шагом будет получение отношения, которое способно генерировать новые логические схемы, эквивалентные исходным. Такое отношение (`isequiv`), представленное в программе 4.11, включает в себя все отношения, использованные ранее при анализе логических схем. В рассматриваемой программе отношение `igate` предназначено для получения логических схем, построенных на элементах с инвертированным выходом `Nor`, `Nand` и `Not`, которые были бы эквивалентны исходным схемам, содержащим только элементы без инвертирования `Or`, `And` и `Link`. Если нужно получить все эквивалентные схемы, то отношение `igate` следует переписать, дополнив соответствующий список так, чтобы он содержал все элементы, включая `Link`.


```

(Link 0) fn 0
(Link 1) fn 1
(Not 0) fn 1
(Not 1) fn 0
(And 0 X) fn 0
(And 1 1) fn 1
(And 1 X) fn Y if
    (And X) fn Y
(Or 0 0) fn 0
(Or 0 X) fn Y if
    (Or X) fn Y
(Or 1 X) fn 1
(Nand X) fn Y if
    (And X) fn Z and
    (Not Z) fn Y
(Nor X) fn Y if
    (Or X) fn Z and
    (Not Z) fn Y
(X) b 1 if
    X ON (0 1)
X b Y if
    1 LESS Y and
    SUM (1 Z Y) and
    x b 1 and
    y b Z and
    APPEND (x y X)
tab (X Y Z x) if
    Z b X and
    (Y Z) fn x
(X Y Z x y) com z if
    tab (X Z X1 Y1) and
    tab (Y x Z1 x1) and
    APPEND (X1 Z1 y1) and
    (y Y1 x1) fn z1 and
    APPEND (y1 (z1) z)
(X Y Z x y) isequiv (X Y z X1 Y1) if
    igate Z and
    igate x and
    igate y and
    (X Y Z x y) equiv (X Y z X1 Y1)
(X Y Z) equiv (X Y x) if
    (forall (X Y x) com y then (X Y Z) com y)
igate X if
    X ON (Not Nor Nand)

```

Приведем пример использования программы 4.11:

```

all (x : x isequiv (2 2 And And And)
(2 2 Nand Nand Nor)
No (more) answers
&.

```

Результат, полученный Пролог-системой после анализа соответствующих таблиц истинности, подтверждает истинность следующего равенства:

$$A \cdot B \cdot C \cdot D = \overline{\overline{A \cdot B} + \overline{C \cdot D}}$$

Доказательство теорем, а также генерация формулировок новых теорем входят в ряд наиболее распространенных задач математической логики и искусственного интеллекта. Каждому изучающему программирование на Прологе будет полезно написать программу, аналогичную рассмотренной в данном разделе, но для такой предметной области, которая больше соответствует его профессиональным интересам. Для читателей, специализирующихся в области электроники, можно порекомендовать в качестве такого упражнения усовершенствовать программу 4.11, дополнив ее отношениями, позволяющими описывать элементы последовательного действия, такие, как триггеры, счетчики и регистры сдвига. Полученная в результате программа должна строить логическую схему по описанию ее работы, представленному в виде спецификаций.

Упражнение 4.11

С помощью программы 4.11 проверить следующие равенства:

a) $A + B + C + D = \overline{\overline{A + B} \cdot \overline{C + D}}$;

б) $A + B \cdot C \cdot D = \overline{\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}}$;

в) $(A + B + C) \cdot D = \overline{\overline{A + B + C} + \overline{D}}$.

Ответы к упражнениям

Упражнение 4.1

а) (TF FT F F); б) (FF); в) (TF FT); г) (TT); д) (F); е) (TT F T FF).

Упражнение 4.2

b = $p \vee q$,

c = $p \vee \sim q$,

e = $\sim p \vee q$,

инверсия от a;

инверсия от n;

инверсия от l.

Упражнение 4.3

a = TRUE (тавтология);

b = $p + q$;

c = $p + q$;

d = \overline{p} ;

e = $p + q$;

f = q ;

g = $p \cdot q + \overline{p} \cdot \overline{q}$;

h = $p \cdot q$;

r = FALSE;

o = $\overline{p \cdot q}$;

n = $\overline{p \cdot q}$;

m = \overline{p} ;

l = $\overline{p \cdot q}$;

k = q ;

j = $\overline{p \cdot q} + \overline{p} \cdot \overline{q}$;

i = $p \cdot q$.

Упражнение 4.4

Функции e и g .

Упражнение 4.5

$p \wedge q = q \wedge p$; $p \vee q = q \vee p$; $(p \wedge q) \wedge r = p \wedge (q \wedge r)$ и т. д.

Упражнение 4.6

а:

1) $A \cdot \bar{B}$; 2) $\overline{A \cdot B}$; 3) $\overline{\overline{A \cdot B \cdot C}}$; 4) $\overline{\overline{A \cdot B \cdot C}}$; 5) $\overline{\overline{A \cdot B \cdot C}}$; 6) $\overline{A \cdot B}$.

б:

1) $\overline{\overline{A + B}}$; 2) $\overline{\overline{A + B}}$; 3) $\overline{\overline{A + B + C}}$; 4) $\overline{\overline{A + B + C}}$; 5) $\overline{\overline{A + B + C}}$; 6) $\overline{A + B}$.

Упражнение 4.7

См. программу 4.11.

Упражнение 4.8

а all (x: (2 2 Nand Nand Nor) com x)

б) all (x: (2 2 And And And) com x)

в) all (x: (1 1 Not Not Nor) com x)

Упражнение 4.9

а) таблицы истинности совпадают;

б) all (x: (2 2 And And And) com x) результаты совпадают, за исключением того, что `fn` выводит на экран символ `X`, означающий «безразличное» состояние, отметим, что таблица для `Or` может быть также получена с помощью следующего запроса:

all (x: (1 1 Link Link Or) com x)

Упражнение 4. 10

а) is ((1 1 Link Link Nand) equiv (1 1 Not Not Or))

б) all (x y: tab (2 Nor x y)).

all (x: (1 1 Not Not And) com x)

Упражнение 4.11

а) is ((2 2 Or Or Or) equiv (2 2 Nor Nor Nand))

б) is ((1 3 Nink And Or) equiv (1 3 Not Nand Nand))

в) is ((3 1 Or Link And) equiv (3 1 Nor Not Nor))

Отметим, что аналогичную информацию и даже несколько большую можно получить с помощью, например, такого запроса:

all (x: x isequiv (3 1 Or Link And))

Ранее уже отмечалась важная роль средств представления знаний в системах искусственного интеллекта и, в частности, в экспертных системах. Конечно, для создания и эксплуатации экспертных систем мощности бытовых ЭВМ недостаточно. Машинам данного типа не хватает, например, внешней памяти, так как для хранения знаний, необходимых при решении даже небольших практических задач, требуется не менее 1000К памяти на гибких дисках.

Для эффективного управления данными, расположенными во внешней памяти, требуются довольно сложные программы. Весь процесс работы с данными может контролироваться специальными программными средствами, называемыми системами управления базами данных (СУБД). Обычно в СУБД входит специальный язык, ориентированный на управление данными. Обращение к программам СУБД может осуществляться с помощью вызовов из программ пользователей, написанных на обычных языках программирования. Следует учитывать, что в большинстве случаев все же целесообразнее использовать на микроЭВМ программы обработки данных, работающие с простыми файловыми системами.

Огромные размеры профессиональных баз данных заставляют предъявлять высокие требования к скорости обработки информации и скорости доступа к данным. Развитие методов параллельной обработки в недалеком будущем обеспечит реализацию алгоритмов быстрого доступа к данным не на программном (как это делается сейчас), а на аппаратном уровне. Скорость доступа к данным на диске при этом должна стать сравнимой со скоростью доступа к устройствам полупроводниковой памяти.

Во многих случаях возникает необходимость обеспечить такой режим работы, когда к одному файлу обращаются несколько пользователей, которые к тому же могут находиться на большом расстоянии от ЭВМ. Такой режим работы называется мультидоступом. В указанном режиме в любой момент времени каждая запись базы данных может быть доступна только одному пользователю — остальные вынуждены ждать. Очевидно, что время такого ожидания пропорционально размерам обрабатываемых записей.

СУБД должны обеспечивать простой и удобный даже для новичков доступ к данным, хранящимся в базе. Важно, что пользователю в этом случае совсем не обязательно знать внутреннюю структуру базы данных. Для удобства ввода информации в настоящее время используются более простые, чем обычная клавиатура устройства, такие, как электронный планшет, мышь, функциональная клавиатура, световое перо, а также системы речевого ввода.

В данном разделе понятие база данных используется в традиционном смысле, особенности же применения различных устройств ввода, а также работа СУБД в многопользовательском режиме здесь не рассматриваются.

5.1. РЕЛЯЦИОННАЯ БАЗА ЗНАНИЙ

Для демонстрации основных положений в настоящем разделе * будет использоваться информация о матчах команд английской футбольной лиги. Эти данные обладают хорошо упорядоченной однородной структурой, т. е. качеством, редко встречающимся в большинстве реальных баз данных, но позволяющим более просто усвоить основные принципы их построения. Между тем, описываемые ниже методы легко могут быть применены для данных других типов.

Информация о матчах представлена в следующем виде:

Дома	В гостях
Команда p w d l f a	w d l f a Очки
Arsn1 12 4 1 1 9 6	2 2 2 6 6 21
(Арсенал)	

Для обозначения названий команд используются сокращения из пяти букв: одинаковый формат названий облегчает их вывод на экран. Для каждой команды, после общего числа сыгранных матчей (p), отдельно для игр на своем поле и в гостях (на поле соперника) перечисляются следующие данные: число выигранных матчей (w), число проигранных (l) и сведенных вничью игр (d), а также число забитых (f) и пропущенных (a) мячей. Наконец, последним в этом ряду стоит количество очков, набранных данной командой в сезоне.

Прежде всего необходимо установить, какие из перечисленных показателей необходимо запоминать в базе данных и в каком

* Здесь и далее рассматриваются два типа программ. Первые из них содержат различные данные, например, результаты футбольных матчей, сведения о наличии материалов на складе и т. д. Программы второго типа содержат описания отношений, предназначенных для разного рода манипуляций с данными, содержащимися в программах первого типа. В системах искусственного интеллекта принято разделять базы данных и базы знаний. В рассматриваемом случае программы первого типа соответствуют базам данных, тогда как программы второго типа можно отнести к базам знаний. — *Прим. пер.*

формате их следует хранить. Очевидно, что число сыгранных матчей равно сумме выигранных, проигранных и сведенных вничью, поэтому данный показатель хранить нет необходимости. Отказываясь от его запоминания в базе данных, мы экономим память. Однако, поскольку этот показатель придется при каждом запросе вычислять, мы тем самым будем проигрывать во времени.

Следует также учесть, что хранение общего числа сыгранных матчей приведет к потерям во времени при еженедельном обновлении информации. Аналогичные выводы могут быть сделаны по поводу показателя «Очки», так как он может быть вычислен по формуле $w \times 3 + d$, где w и d — общее число выигранных и сведенных вничью матчей. Для начала договоримся не запоминать указанные два показателя, а вычислять их при поступлении запросов.

Рассмотренные выше показатели каждой команды объединяются в список, состоящий из двух аналогичных друг другу подсписков, первый из которых состоит из показателей для игр, проведенных дома, а второй — для матчей на чужих стадионах. Соответствие между названиями команд и списками показателей устанавливается с помощью отношения `form` (сформированная строка-показателей), например, следующим образом:

`Arsnl form ((4 1 1 9 6) (2 2 2 6 6)).`

Все команды лиги играют только с командами своей подгруппы. Поэтому можно для каждой подгруппы использовать отдельную программу. Такие программы будут работать быстрее, чем одна общая для всей лиги. В начале каждого сезона необходимо проводить инициализацию таблиц для каждой подгруппы. Для этой цели можно использовать отношение `setup` (установить-начальные-значения):

```
X setup if
  (team X) is-told and
  (X form (0 0 0 0 0) (0 0 0 0 0)) add
```

Названия команд могут вводиться в произвольном порядке и, поскольку применяется реляционная модель хранения, пользователю не нужно знать, где именно в базе данных размещается информация о каждой команде. Однако можно обнаружить, что во время инициализации списков подгрупп информация может каждый раз записываться на новое место.

Пользователь может ввести названия команд подгруппы следующим образом:

```
all (: X setup)
[определить все (: X установить-начальные-значения)]
Team X ? ans Mputd
[команда X ? ответ Манчестер-Юнайтед]
Team X ? ans Lpool
```

[команда X ? ответ Ливерпуль]
 Team X ? ans Chlse
 [команда X ? ответ Челси]
 Team X ? just Arsnl
 [команда X ? последний ответ Арсенал]
 No (more) answers
 [Ответов (больше) нет]

В этом случае по команде list form будет выдана следующая информация, представляющая собой начальный вариант таблицы первенства:

Mnutd form ((Ø Ø Ø Ø Ø) (Ø Ø Ø Ø Ø))
 Lpool form ((Ø Ø Ø Ø Ø) (Ø Ø Ø Ø Ø))
 Chlse form ((Ø Ø Ø Ø Ø) (Ø Ø Ø Ø Ø))
 Arsnl form ((Ø Ø Ø Ø Ø) (Ø Ø Ø Ø Ø)) ...

После того, как подгруппа сформирована, отношение setup может быть удалено.

При вводе могут быть допущены ошибки или появится какая-либо причина для внесения исправлений в таблицу. Можно было бы воспользоваться командой редактирования для формирования таблицы, но лучше применять для этой цели специальные средства, например, приведенное ниже отношение correct (исправить):

X correct if
 (team X form ((Y Z x y z) (X1 Y1 Z1 x1 y1))) is-told and
 (X form ((Y Z x y z) (X1 Y1 Z1 x1 y1))) add and
 (X form z1) delete

Следующий диалог демонстрирует использование данного отношения:

all (X: X correct)
 [определить все (X: X изменить)]
 Team X form ((Y Z x y z) (X1 Y1 Z1 x1 y1)) ? just Mnutd 6 Ø
 Ø 15 Ø 5 1 Ø 15 4
 [команда X формировать ((Y Z x y z) (X1 Y1 Z1 x1 y1)) ? последний-ответ Манчестер-Юнайтед 6 Ø Ø 15 Ø 5 1 Ø 15 4]
 Mnutd
 [Манчестер-Юнайтед]
 No (more) answers
 [Ответов (больше) нет]

Для иллюстрации материалов данного раздела была взята информация о положении команд первой подгруппы на 13 октября 1985 г., приведенная полностью в табл. 5.1.

Вариант заполнения турнирной таблицы

Mnutd	form ((6 0 0 15 0) (5 1 0 15 4))
Lpool	form ((6 0 0 17 2) (1 3 2 10 11))
Chlse	form ((6 0 0 12 2) (1 3 2 5 9))
Arsnl	form ((4 1 1 9 6) (2 2 2 6 6))
Shelfw	form ((2 3 1 11 13) (4 0 2 8 7))
Evrtn	form ((4 1 1 13 5) (2 1 3 8 9))
Wtfrd	form ((6 0 0 19 4) (0 1 5 7 17))
Ncstl	form ((4 1 1 14 6) (1 3 2 6 12))
Q-P-R	form ((5 0 1 11 4) (1 0 5 4 13))
Spurs	form ((4 0 1 18 4) (1 2 3 5 9))
Wsthm	form ((3 2 1 12 6) (1 3 2 7 9))
Nottf	form ((3 0 3 9 6) (2 1 3 9 10))
Bmham	form ((4 1 1 7 4) (1 0 4 3 12))
Luton	form ((2 4 0 8 5) (1 2 3 7 11))
Cvtry	form ((3 1 2 13 7) (0 4 2 5 9))
Avlla	form ((1 3 2 8 8) (2 2 2 8 7))
Shmtn	form ((2 4 0 9 4) (0 1 5 4 10))
Oxfrd	form ((2 3 1 10 5) (0 2 5 8 21))
Leics	form ((2 4 1 12 12) (0 1 5 2 15))
Mncty	form ((1 2 3 6 11) (1 1 4 6 11))
lpswh	form ((1 2 3 4 8) (1 0 5 3 12))
W-B-A	form ((0 2 4 6 15) (0 1 5 4 19))

Далее рассмотрим отношения, используемые для периодического обновления данных турнирной таблицы. Сначала покажем, как данная операция может быть реализована с помощью встроенных средств Пролога. При выполнении операции обновления должна быть доступна каждая запись таблицы. Вот пример запроса, позволяющего получить одну из записей, относящуюся к команде, обозначенной как Luton:

```
all (x: x form ((y | z) X) and y LESS 4)
((2 4 0 8 5) (1 2 3 7 11))
```

Для того чтобы определить команды, выигравшие дома более четырех матчей, можно использовать следующий запрос:

```
all (x y: x form (Y | z) X) and y LESS 4)
Mnutd 6
Lpool 6
Chlse 6
Q-P-R 5
No (more) answers
```

Следующий пример демонстрирует возможность использования средств Пролога для составления списков:

```
all (x y z: x form (X (z z z | Z)) and
y form (Y (z z z | Z1)) and
x LESS y)
Arsnl AVlla 2
```

Данный запрос позволяет определить команды, имеющие одинаковое число выигранных, проигранных и сведенных вничью матчей. Если читатель не смог сразу вспомнить назначение условия `x LESS y`, ему следует попробовать выполнить тот же запрос без него.

Упражнение 5.1

Составьте запросы для получения:

- 1) числа игр, проведенных данной командой дома;
- 2) числа игр, проведенных данной командой на чужих стадионах;
- 3) общего числа игр, проведенных данной командой;
- 4) числа очков, набранных данной командой;
- 5) списка команд, имеющих число забитых мячей больше, чем у команды Вест Хэм (Wsthm).

Упражнение 5.2

Составьте отношения, с помощью которых можно найти:

- 1) название команды, оказавшейся наиболее результативной в играх, проведенных дома;
- 2) название команды, оказавшейся наиболее результативной в играх, проведенных на чужих полях;
- 3) название команды, забившей наибольшее число мячей во всех играх.

С помощью предложенных выше упражнений читатель смог убедиться в том, что работать с базой данных, пользуясь только встроенными средствами микроПролога, не очень удобно.

Запросы получаются длинными, к тому же составлять их не так просто. Доступ к информации хорошо организованной базы данных должен быть простым и удобным для пользователя. Отношения, определенные в программе 5.1, позволяют легко делать запросы для получения информации, аналогичной той, которая требовалась в упражнениях 5.1 и 5.2.

Программа 5.1

```
X htop Y if
  hgoals Z and
  Y max Z and
  form (X ((x y z Y X1) Y1))
X stop Y if
  agoals Z and
  Y max Z and
  form (X (x (y z X1 Y Y1)))
X alltop Y if
  allgoals Z and
  Y max Z and
  form (X ((x y z X1 Y1) (Z1 x1 y1 z1 X2))) and
  SUM (X1 z1 Y)
hgoals X if
  X isall (X: form (Y ((Z x y X z) X1)))
agoals X if
  X isall (X: form (Y (Z (x y z X X1))))
allgoals X if
  X isall (X: form (Y ((Z x y z X1) (Y1 Z1 x1 y1 z1))) and SUM (z y1 X))
X max Y if
  X ON Y and
  (forall Z ON Y and not Z EQ X then Z LESS X) and
  /
```

Приведем краткую характеристику функций, реализуемых отношениями программы 5.1:

htop — поиск команды, оказавшейся наиболее результативной в играх на своем поле;

atop — поиск команды, забившей наибольшее число голов в играх на чужих полях;

alltop — поиск наиболее результативной команды;

hgoals — получение списка, каждый элемент которого представляет собой общее число мячей, забитых командой в играх, проведенных дома;

agoals — получение списка, каждый элемент которого представляет собой общее число мячей, забитых командой в матчах на чужих полях;

allgoals — получение списка, каждый элемент которого представляет собой общее число забитых командой мячей;

max — поиск максимального элемента в произвольном списке.

К программе 5.1 можно обратиться, например, следующим образом:

```
all (x y: x htop y)
```

```
[определить все (x y: x забито-наибольшее-число-мячей y)]
```

```
Wtfrd 19
```

```
[Вутфорд 19]
```

```
No (more) answers
```

```
[Ответов (больше) нет]
```

Упражнение 5.3

Напишите запросы, которые позволили бы определить команду, имеющую:

1) наибольшее число забитых мячей как дома, так и в гостях;

2) наибольшее число забитых мячей во всех играх и в играх дома;

3) наибольшее число забитых мячей во всех играх и в играх на чужих полях.

В зависимости от дальнейшего использования базы данных программист может дополнить программу другими отношениями. Если понадобятся сведения о максимальных значениях других показателей, то соответствующие отношения могут быть составлены по аналогии с приведенными в программе 5.1. Однако в том случае, когда нужно получать показатели по какой-либо одной команде, в качестве образца можно взять программу 5.2. Если же необходимы оба типа информации, указанные программы могут быть объединены в одну.

Достоинством Пролога является то, что программы могут присоединяться к другим уже загруженным ранее в рабочую область программам, без перегрузки последних. Однако в таких случаях необходимо избегать совпадения названий вводимых отношений с ранее размещенными в рабочей области.

Когда разработка программ полностью завершена, их следует оформить в виде модулей.

Показатели команд разных подгрупп должны храниться в отдельных модулях таким образом, чтобы данные могли быть загружены в рабочую область, подвергнуты необходимой обработке и затем удалены из нее без изменения обрабатывающих модулей. Другое преимущество такого способа обработки заключается в том, что модули, содержащие только информацию о командах, короче тех, которые включают в себя еще и обрабатывающие программы.

Поэтому для их записи на внешние запоминающие устройства требуется меньше времени.

Программа 5.2

```
(X Y) match ((X|Z) (Y|x))
(X Y) match ((Z|x) (y|z)) if
  (X Y) match (x z)
result (won drew lost for against)
X hrec (Y Z) if
  result x and
  X form (y z) and
  (Y Z) match (x y)
X arec (Y Z) if
  result x and
  X form (y z) and
  (Y Z) match (x z)
X rec (Y Z) if
  X hrec (Y x) and
  X arec (Y y) and
  SUM (x y Z)
```

В программе 5.2 отношение hrec позволяет получать показатели по играм, проведенным дома, arec — по играм на чужих полях и, наконец, rec — по всем играм. Ниже приведен пример использования данной программы:

```
all (x: Mnutd rec x)
[определить все (x : Манчестер-Юнайтед все-игры x)]
(won 11)
[побед 11]
(drew 1)
```

[ничьих 1]
(lost 0)
[проигрышей 0]
(for 30)
[забито 30]
(against 4)
[пропущено 4]
No (more) answers
[Ответов (больше) нет]

Вот другой пример использования программы 5.2, когда требуется получить для всех команд значения одного показателя:

all (x scored y goals: x rec (for y))
[x забил y голов : x все-игры (забито y)]
Mnutd scored 30 goals
[Манчестер-Юнайтед забил 30 голов]
Lpool scored 27 goals...
[Ливерпуль забил 27 голов]
и т. д.

Упражнение 5.4

С помощью программы 5.2 напишите запросы, позволяющие, определить:

- 1) все команды, забившие больше 20 голов;
- 2) все команды, забившие в играх дома голов больше, чем Эвертон (Evrtn);
- 3) все команды, забившие одинаковое число голов (требуется также вывести и число голов);
- 4) все команды, которые больше игр сыграли вничью, чем проиграли;
- 5) все команды, проигравшие больше матчей дома, чем Арсенал (Arsnl) выиграл на чужих стадионах.

5.2. МОДИФИКАЦИЯ ЗАПИСЕЙ

Для базы данных, содержащей информацию о футбольной турнирной таблице, необходимы средства для регулярного ее обновления. Таким средством может служить программа 5.3, позволяющая обновлять турнирные показатели, вводя информацию о сыгранных матчах с помощью строк типа:

(название команды хозяев) Н А (название команды гостей),

где Н и А — числа голов, соответственно забитых и пропущенных хозяевами.

```

X adj (Y Z x y) if
  (Team X score Z x Team Y) is-told and
  (either (either x LESS Z and home y and / or Z LESS x and away y
    and /) and / or draw y and /) and
  X adj1 (Y Z x y) and
  X adj2 (Y Z x y)
X adj1 (Y Z x (y z X1)) if
  (X form((Y1 Z1 x1 y1 z1) X2)) delete and
  SUM (Z y1 Y2) and
  SUM (x z1 Z2) and
  SUM (y Y1 x2) and
  SUM (z Z1 y2) and
  SUM (X1 x1 z2) and
  (X form ((x2 y2 z2 Y2 Z2) X2)) add and
  /
X adj2 (Y Z x (y z X1))if
  (Y form (Y1 (Z1 x1 y1 z1 X2))) delete and
  SUM (Z X2 Y2) and
  SUM (x z1 Z2) and
  SUM (y y1 x2) and
  SUM (z x1 y2) and
  SUM (X1 Z1 z2) and
  (Y form (Y1 (z2 y2 z2 Z2 Y2))) add and
  /
home (1 0 0)
draw (0 1 0)
away (0 0 1)

```

Для работы программы 5.3 необходим модуль TOLD. Ниже приведен возможный вариант диалога с указанной программой:

```

all (: x adj y)
[определить все (: x текущие-изменения y)]
Team X score Y Z Team x ? ans Arsnl 1 0 Ipswh
[команда X счет Y Z команда x? ответ Арсенал 1 0 Ипсвич]
Team X score Y Z Team x ? just Evrtn 4 1 Wtfrd
[команда X счет Y Z команда x? последний-ответ Эвертон 4 1
Уитфорд]
No (more) answers
[Ответов (больше) нет]

```

Если вывести после этого на экран турнирную таблицу (т. е. все элементы отношения form), то для указанных четырех команд строки изменятся следующим образом:

```

Arsnl form ((5 1 1 1 0 6) (2 2 2 6 6))
Ipswh form ((1 2 3 4 8) (1 0 6 3 13))
Evrtn form ((5 1 1 17 6) (2 1 3 8 9))
Wtfrd form ((6 0 0 19 4) (0 1 6 8 21))

```

С помощью отношения add новые предложения добавляются в конец списка предложений отношения form. Однако для поль-

зователя безразлично, в каком именно месте размещена информация о каждой команде. Отношение `adj` в программе 5.3 в зависимости от результата игры формирует строку параметров следующим образом: если выиграл хозяева поля, в строку записывается последовательность (1 0 0), если выиграл гости, то передается (0 0 1), тогда как в случае ничьей формируется последовательность (0 1 0). Далее указанная строка передается двум вспомогательным отношениям `adj1` (текущие-изменения-1) и `adj2` (текущие-изменения-2). Первое из них изменяет запись, относящуюся к команде хозяев, прибавляя число голов, забитых хозяевам, и число голов, забитых гостями, к значениям соответствующих показателей.

Второе отношение выполняет аналогичную операцию, но для команды гостей. И то и другое отношение уничтожают записи со старыми показателями и добавляют соответствующие обновленные записи. Хотя обновление таблицы производится каждую неделю, за этот период накапливается много информации, что вызывает многократное выполнение операций поиска и удаления старых, а также добавления новых записей. Поэтому программа работает достаточно долго.

Для повышения эффективности работы системы программы обработки следует оформлять в виде самостоятельных модулей и хранить их отдельно от программ, содержащих данные. Это можно сделать с помощью системного модуля Пролога `MODULES`. Сначала его надо загрузить в оперативную память, а затем либо ввести текст программы 5.3 с клавиатуры, либо, если этот текст был предварительно записан на магнитофон, с магнитофонной кассеты.

После того, как вы удостоверитесь, что в программе содержатся только предложения `adj`, `adj1`, `adj2`, `home`, `away` и `draw`, введите еще одно дополнительное предложение:

`Module (adjust-mod (adj) (form add delete is-told))`

Данное предложение содержит два списка. В первом списке, называемом списком экспорта, перечисляются имена тех отношений, которые должны стать доступными для внешних вызовов. В приведенном примере этот список состоит из одного элемента — `adj`, но при желании его можно было бы дополнить элементами `adj1` и `adj2`. Второй список, называемый списком импорта, содержит имена отношений, необходимых для работы программы 5.3. Отношение `form` берется из программ пользователя, содержащих данные (из баз данных), отношение `is-told` — из системного модуля `TOLD` и, наконец, отношения `add` и `delete` — из системного модуля `SIMPLE`. Отношения `LESS` и `SUM` в список импорта включать не требуется, так как они являются стандартными отношениями Пролога и доступны для всех модулей.

Теперь необходимо ввести с клавиатуры предложение «`тгар ADJUST`», но перед тем, как нажать клавишу «ВВОД», нужно

включить магнитофон для записи модуля. После завершения записи, если не было обнаружено ошибок, на экране должно появиться сообщение о том, что рабочая область очищена («Workspace now clear»).

Для того чтобы обратиться к модулю ADJUST, нужно перемотать кассету назад и затем загрузить его. К информации о турнирных таблицах подгрупп, заранее записанных на кассету, можно обратиться с помощью запроса следующего вида:

all (: x adj y)

После ввода указанного запроса на экране должно появиться уже знакомое читателю сообщение, приглашающее вводить данные. Затем соответствующая таблица будет обновлена с помощью введенных данных и перезаписана с учетом изменений. Данная процедура может быть выполнена для турнирной таблицы каждой подгруппы, т. е. для любого файла, содержащего отношение типа

X form ((Y Z x y z) (X1 Y1 Z1 x1 y1))

5.3. СРЕДСТВА МОДИФИКАЦИИ ЗАПИСЕЙ

Несмотря на то, что модуль ADJUST был создан для обновления данных футбольной таблицы, он может быть использован для модификации файлов, содержащих информацию другого вида, но имеющих структуру, аналогичную рассмотренной выше. Применению указанного модуля в качестве универсального препятствуют следующие три момента.

1. В каждом предложении adj1 и adj2 связь со списком данных задается с помощью отношения form. Хотя имя этого отношения в данном случае достаточно удобно, следует все-таки использовать более общие, универсальные имена отношений, такие, как stat или data. Выбранное имя должно быть включено в список импорта модуля ADJUST. Следует избегать длинных списков импорта и стараться пользоваться одним универсальным отношением. Пусть, например, в качестве имени универсального отношения выбрано data. Тогда для того, чтобы можно было импортировать имя выбранного отношения из прикладной программы, в нее необходимо добавить следующий фрагмент:

```
X data Y if  
X form Y
```

Следует отметить, что при использовании универсальных имен наглядность программирования до некоторой степени теряется. Так, для пользователя описанной выше программы более удобно было бы включить в список импорта отношение form. Поэтому, чтобы сохранить наглядность текстов программ, необходимо

стараться избегать использования унифицированных имен отношений внутри программ.

2. В рассмотренном примере регулярно обновляемая информация состоит из двух подписков, по пять элементов в каждом. Более универсальная программа должна уметь обрабатывать списки из произвольного числа подписков любой длины.

3. Запрос типа «Team X score Z x Team Y» удобен для обработки данных турнирных таблиц, но явно не подходит для других приложений.

Программа 5.4 обеспечивает работу с базами данных более общего типа. При этом пользователь должен указать список всех используемых полей, всех ключей и всех отношений, связывающих ключи с полями.

Программа 5.4

```
begin X if
  (X list fields) is-told and
  (field X) add and
  Y zero X and
  (zero-field Y) add
begin X if
  (X list keys) is-told and
  (key X) add
begin (X Y Z) if
  (Y state relation name) is-told and
  zero-field Z and
  key x and
  X ON x and
  (X Y Z) add
(0) zero (X)
(0 X) zero (Y|Z) if
  X zero Z
```

Отношение zero возвращает список, содержащий число нулей, равное числу элементов заданного списка. Например:

```
which (x: x zero (a b c d e))
(0 0 0 0 0)
No (more) answers
```

Первое предложение отношения begin запрашивает у пользователя имена всех полей для обработки и затем добавляет к программе, находящейся в рабочей области, два предложения:

```
field (a b c d e)
zero field (0 0 0 0 0)
```


где a, b, c, d и e — имена полей, названные пользователем, а zero field представляет собой список с числом нулей, равным числу названных полей.

С помощью предложений отношения begin пользователь вводит значения ключей. После этого к программе, находящейся в рабочей области, добавляется предложение: «key (ABCD)», где A, B, C и D — значения ключей.

В третьем предложении запрашивается имя отношения и в рабочую область записывается по одному предложению с введенным именем отношения для каждого ранее введенного ключа. В рассматриваемом примере будут добавлены предложения следующего вида:

```
A rel (Ø Ø Ø Ø Ø)
B rel (Ø Ø Ø Ø Ø)
и т. д.
```

Здесь rel — имя отношения, выбранное пользователем. Следует отметить, что в выражении «(X Y Z) add» из третьего правила программы 5.4 символ Y обозначает имя введенного отношения.

Таким образом, пользователь добавляет к своей программе отношение, определенное им самим.

Возможность использования имен отношений в качестве переменных представляет собой одно из мощных средств Пролога. В рассматриваемом случае данная возможность позволяет пользователю создать свой язык управления базой данных. Следует отметить, что для правильной работы программы 5.4 число полей и число ключей не должно быть меньше двух. Для каждого списка полей и каждого списка ключей может быть задано несколько разных имен отношений. Ниже приведен пример работы программы 5.4:

```
all (x: begin x)
[определить все (x: инициализация x)]
X list all fields ? just (won drew lost for against)
[X перечислить все поля? последний-ответ (выиграно вничью
проиграно забито пропущено)]
(won drew lost for against)
[(выиграно вничью проиграно забито пропущено)]
X list all keys ? just (Mnuted Lpool Chlse Arsnl)
[X перечислить все ключи? последний-ответ (Манчестер-
Юнайтед Ливерпуль Челси Арсенал)]
(Mnuted Lpool Chlse Arsnl)
[(Манчестер-Юнайтед Ливерпуль Челси Арсенал)]
X state relation name ? just form
[X установить имя отношения? последний-ответ form]
```

```

(Mnutd form (Ø Ø Ø Ø Ø))
(Lpool form (Ø Ø Ø Ø Ø))
(Chlse form (Ø Ø Ø Ø Ø))
(Arsnl form (Ø Ø Ø Ø Ø))
No (more) answers
[Ответов (больше) нет]

```

Приняв информацию от пользователя, программа повторяет введенные им списки имен, полей и ключей, а затем использует их при формировании предложений типа «(ключ) (отношение) (Ø Ø Ø...)». Для того чтобы избежать повторов Пролог-системой вводимой информации, необходимо вместо «all (x : begin x)» использовать «all (: begin x)».

В результате работы программы к базе данных, находящейся в рабочей области, будут добавлены следующие предложения:

```

field (won drew lost for against)
zero-field (Ø Ø Ø Ø Ø)
key (Mnutd Lpool Chlse Arsnl)
(Mnutd form (Ø Ø Ø Ø Ø))
(Lpool form (Ø Ø Ø Ø Ø))
(Chlse form (Ø Ø Ø Ø Ø))
(Arsnl form (Ø Ø Ø Ø Ø))

```

Как было указано выше, можно определить несколько имен отношений, хотя обычно этого не требуется. Примером такого случая, когда для одного отношения могут понадобиться два различных имени, является вариант представления турнирной таблицы, когда списки с результатами игр, проведенных дома, хранятся отдельно от списков с результатами матчей на чужих полях. В качестве имен отношений для такого способа хранения информации могут быть выбраны, например, `hform` и `aform` для каждой из двух указанных частей базы данных соответственно. В этом случае с помощью двух отношений устанавливается связь между именем ключа (названием команды) и соответствующими двумя списками. В рассмотренном выше примере был определен входной формат, в котором для каждой команды обеспечивался ввод следующих полей: `won` (число выигранных матчей), `lost` (число проигранных), `for` (забитые мячи) и `against` (пропущенные мячи). Вводимые данные помещались в эти списки, относящиеся к одной и той же команде.

Однако можно предусмотреть отдельный ввод в эти два списка:

```

all (: begin x)
[определить все (: инициализация x)]
X list all fields ? just (won drew lost for against)

```

[X перечислить все поля? (выиграно вничью проиграно забито пропущено)]

X list all keys ? just (Mnutd Lpool Chlse Arsnl)

[X перечислить все ключи? последний-ответ (Манчестер-Юнайтед, Ливерпуль, Челси, Арсенал)]

X state relation name ? ans hform

[X установить имя отношения? ответ hform]

X state relation name ? just aform

[X установить имя отношения? последний-ответ aform]

Здесь Пролог-система не повторяет вводных пользователем строк, так как запрос начинается с предложения «all (: begin x)». В результате работы программы 5.4 к рабочей области будут добавлены следующие предложения:

field (won drew lost for against)

zero-field (Ø Ø Ø Ø Ø)

key (Mnutd Lpool Chlse Arsnl)

(Mnutd hform (Ø Ø Ø Ø Ø))

(Lpool hform (Ø Ø Ø Ø Ø))

(Chlse hform (Ø Ø Ø Ø Ø))

(Arsnl hform (Ø Ø Ø Ø Ø))

(Mnutd aform (Ø Ø Ø Ø Ø))

(Lpool aform (Ø Ø Ø Ø Ø))

(Chlse aform (Ø Ø Ø Ø Ø))

(Arsnl aform (Ø Ø Ø Ø Ø))

Программа 5.4 может быть использована для создания баз данных различного типа. Таким образом, по сравнению с ранее рассмотренной программой, основанной на отношении setup, ее можно считать более универсальной. Прежде чем продолжить изучение книги, читателю рекомендуется выполнить упражнения 5.5 и 5.6.

Упражнение 5.5

С помощью программы 5.4 создайте базу данных для трех фирм — Мак Ду (McDoo), Фи (Fee) и Хоум (Home), торгующих строительными материалами: песком (sand), цементом (cement), кирпичом (bricks), бетонными блоками (blocks), гравием (gravel), штукатуркой (plaster) и краской (paint). Начальные значения полей должны быть нулевыми.

Упражнение 5.6

С помощью составленной в предыдущем упражнении программы создать базу данных, содержащую информацию о покупке и продаже указанных в том же упражнении строительных мате-

риалов (сформировать отношения, аналогичные рассмотренным выше $hform$ и $aform$ для обработки результатов игр, проведенных дома, и матчей на чужих полях, соответственно).

5.4. МИНИАТЮРНАЯ СУБД

Теперь, используя идеи, описанные в предыдущих разделах, построим небольшую систему управления базами данных. Хотя по сравнению с настоящими наша СУБД будет намного проще, но ее использование поможет читателю ближе познакомиться с проблемами, возникающими при создании больших СУБД. При разработке СУБД следует учитывать, что система должна быть удобна как для обучения работе с ней, так и в процессе самой работы. В системах, предназначенных для обработки данных только одного типа, все виды запросов от пользователя легко предусмотреть заранее. В отличие от указанных примитивных систем все запросы к универсальным СУБД заранее предвидеть нельзя. Поэтому при построении системы необходимо учесть возможность появления непредусмотренных запросов.

Можно выделить три режима работы СУБД: инициализацию, модификацию и чтение информации. В режиме инициализации база данных создается. При этом определяются типы данных, предназначенных для хранения, программные средства доступа к информации и общий объем базы данных. При модификации базы данных пользователь может исправлять, удалять или добавлять отдельные поля, ключи и записи. В режиме чтения (или доступа) пользователи получают информацию из базы данных. При этом доступ может осуществляться по значению ключей и отдельных полей записей. В ряде случаев различные режимы могут быть совмещены.

Например, инициализация базы может выполняться с вводом (модификацией) данных. Часто объединяют также режимы чтения и модификации.

Как было указано ранее, система программирования микроПролог может рассматриваться как СУБД реляционного типа. Пользуясь данной системой как своего рода оболочкой, опытные программисты могут создавать разнообразные СУБД для пользователей, не обладающих навыками работы на языке Пролог. При создании таких СУБД ряд функций микроПролога будет дублироваться. Так, например, с помощью встроенных средств рассматриваемой системы можно отредактировать предложение некоторого отношения, зная номер данного предложения.

Однако в большой программе отношение может содержать сотни предложений, и поиск нужного среди них в таком случае становится для неопытного пользователя довольно затрудни-

тельным. Создавая нашу СУБД, мы будем пытаться сделать указанный процесс исправления записей более удобным и простым для пользователей.

5.5. ИНИЦИАЛИЗАЦИЯ БАЗЫ ДАННЫХ

Прежде всего следует написать процедуру, инициализирующую основные поля, названия ключей, а также позволяющую определить вид отношения, связывающего ключи с полями. Кроме этого, процедура инициализации должна в случае необходимости обеспечить добавление новых полей, ключей и даже отношений. При этом не следует забывать о простоте доступа к информации базы данных.

Модернизируем программу 5.4 так, чтобы можно было с ее помощью осуществлять инициализацию базы данных. Текст новой программы 5.5 приводится ниже.

Программа 5.5

```
begin X if
  (X list fields) is-told and
  (fields X) add and
  Y zero X and
  (zeroes Y) add
begin X if
  (X list keys) is-told and
  (keys X) add
begin X if
  (X list relations) is-told and
  (rels X) add
begin (X Y Z) if
  keys x and
  X ON x and
  rels y and
  Y ON y and
  zeroes Z and
  (X Y Z) add
(0) zero (X)
(0|X) zero (Y|Z) if
  X zero z
Module (begin-mod (begin zero) (add is-told
rels))
```

Данную программу (с последней строкой QN fields keys zeroes rels)) вместо rels))) следует оформить в виде отдельного модуля, для чего необходимо загрузить модель MODULES, после чего выполнить команду «wgar BEGIN». Последние четыре отношения из включенных в список импорта обеспечивают генерацию произвольной базы данных.

После этого рабочая область будет очищена. Затем полученный модуль нужно заново загрузить, выполнив команду «load BEGIN». Теперь можно создавать базу данных, но сначала

с помощью команды «KILL modules-mod» необходимо очистить память.

В качестве примера создадим базу данных, содержащую информацию о состоянии хозяйства фермеров: Хилл (Hill), Дейл (Dale), Мндоуз (Meadows) и Рокс (Rocks). В эту базу данных будут вводиться сведения о численности птицы и поголовья скота этих фермеров. Ниже приводится пример диалога при инициализации указанной базы данных:

```
all (x: begin x)
[определить все (x: инициализировать 'x')]
X list all fields ? just (cows pigs sheep fowl)
[X перечислить все поля? последний-ответ (коровы свиньи
овцы птица)]
(cows pigs sheep fowl)
[(коровы свиньи овцы птица)]
X list all keys ? just (Hill Dale Meadows Rocks)
[X перечислить все ключи? последний-ответ (Хилл Дейл Мн
доуз Рокс)]
(Hill Dale Meadows Rocks)
[(Хилл Дейл Мндоуз Рокс)]
X list all relations ? just (in out)
[перечислить все отношения? последний-ответ (in out)]
(in out)
(Hill in (Ø Ø Ø Ø))
(Hill out (Ø Ø Ø Ø))
(Dale in (Ø Ø Ø Ø))
и т. д.
```

В результате выполнения данной программы будет создана следующая база данных:

```
list all
fields (cows pigs sheep fowl)
zeroes (Ø Ø Ø Ø)
keys (Hill Dale Meadows Rocks)
in iels out
Hill in (Ø Ø Ø Ø)
  Meadows in (Ø Ø Ø Ø)
  Hill out (Ø Ø Ø Ø)
  Dale out (Ø Ø Ø Ø)
  Meadows out (Ø Ø Ø Ø)
```

Итак, с помощью модуля BEGIN сгенерирована база данных с нулевыми начальными значениями полей. Теперь перейдем к созданию модуля, позволяющего модифицировать эту базу данных. Но сначала остановимся на некоторых особенностях процесса модификации.

1. Отношения `in` и `out` могут быть определены иначе, чем было предложено раньше, — с помощью команды микроПролога `assert`. В этом случае при вводе данных каждый ключ и каждое поле должны набираться на клавиатуре полностью, например, как показано ниже:

```
accept in
in (Hill (Ø Ø Ø Ø))
in (Dale (Ø Ø Ø Ø))
и т. д.
```

Таким образом, прежний вариант позволяет избежать лишней работы по вводу данных. Хотя значения полей рано или поздно должны быть введены, при генерации базы данных для большей части полей они часто бывают еще не известны. Модуль `BEGIN` обеспечивает ввод нулевых значений для неопределенных данных и, таким образом, формирует базу данных полностью.

2. Первые четыре предложения программы 5.5 служат для определения структуры записей базы данных, а также способов доступа к этим записям. С помощью этих предложений пользователь может самостоятельно добавлять или уничтожать отдельные ключи, поля или отношения. При этом от него не требуется знать, где в базе данных размещены указанные элементы.

3. Модуль, полученный из программ 5.5, позволяет создавать различные базы данных, в которых для одного ключа устанавливается связь с определенным набором полей с помощью отношения, задаваемого пользователем.

4. С помощью обобщенной формы модуля `BEGIN` пользователь может определять отношения, наиболее подходящие для его целей. Так, например, два отношения `in` и `out`, заданные пользователем в приведенном выше примере, предназначены для внесения в базу данных изменений об увеличении или, соответственно, уменьшении численности птицы и поголовья скота.

5. Отношение `rels`, описанное в программе 5.5, аналогично во многом встроенному отношению микроПролога `dict`, но в отличие от последнего содержит только имена отношений без указания ключей и полей.

Упражнение 5.7

С помощью модуля `BEGIN` постройте базы данных, содержащие следующую информацию:

- 1) отметки студентов по различным предметам;
- 2) имена игроков различных команд;
- 3) число книг в вашей библиотеке по каждой тематике отдельно;

- 4) число растений в вашем саду (по каждому виду отдельно);
- 5) номера автобусных маршрутов (по каждой улице отдельно).

При выполнении указанных упражнений следует ограничиться небольшим количеством ключей и полей и избегать громоздких примеров. Базы данных должны содержать лишь начальные нулевые значения.

5.6. ОБНОВЛЕНИЕ БАЗЫ ДАННЫХ

В процессе работы с базой данных может возникнуть необходимость в ее модификации, т. е. в добавлении новых ключей, полей или отношений. Указанные операции можно выполнить с помощью встроенных отношений `add`, `delete` и `edit`, но при этом необходимо располагать полным сведениями о содержимом базы данных. В связи с этим такой способ скорее всего будет сопровождаться многочисленными ошибками. Нам же необходима программа, которая после получения от пользователя соответствующей команды сможет выполнить все операции по модификации базы данных автоматически.

Сначала рассмотрим случай, когда либо новая информация добавляется в базу данных, либо производится модификация данных, уже имеющихся в базе. Варианты, возможные в этом случае, приведены в табл. 5.2.

Таблица 5.2

Варианты типов запросов
для изменения базы данных

Поле	Ключ	Отношение
Нет	Нет	Нет
Нет	Нет	Да
Нет	Да	Нет
Да	Да	Да
Да	Нет	Нет
Да	Нет	Да
Да	Да	Нет
Да	Да	Да

Каждая строка табл. 5.2 соответствует какому-либо варианту запроса. Если запрос изменяет ключ, поле или отношение, то в соответствующей его позиции стоит «да». В противном случае эта позиция помечается словом «нет». Можно создать отношение, которое изменяло бы базу данных по описанным выше запросам, но операции модификации в этом случае выполнялись бы очень медленно. Разобьем все процедуры изменения базы данных на два класса: к первому отнесем те из них, в которых изменению подлежат уже существующие поля базы данных, а ко второму — все остальные. В соответствии с этими двумя классами создадим два отношения: `addf` (для добавления нового поля) и `update` (для добавления новых ключей, отношений, а также модификации существующих записей). Указанные отношения представлены в программе 5.6. С помощью данной программы можно модифицировать любую базу данных, созданную с использованием модуля `BEGIN`.

Программа 5.8

```

addf X if
    (fields Y) delete and
    (X new field) is-told and
    APPEND (Y (X) Z) and
    (fields Z) add and
    (zeroes x) delete and
    APPEND (x (0) y) and
    (zeroes y) add and
    /
update (X Y Z) if
    (key X relation Y and new data Z) is-told and
    updatel (X Y Z)
updatel (X Y Z) if
    X key and
    (either Y rel and / or (rels x) delete and APPEND (x (Y) y) and
    (rels y) add and /) and
    (X Y x) delete and
    (X Y Z) add and
    /
updatel (X Y Z) if
    not X key and
    (X Y Z) add and
    (keys x) delete and
    APPEND (x (X) y) and
    (keys y) add and
    (either Y rel and / or (rels z) delete and APPEND (z (Y) X1) and
    (rels X1) add and /) and
    /
X key if
    keys Y and
    X ON Y
X rel if
    rels Y and
    X ON Y
X field if
    fields Y and
    X ON Y
X elim Y if
    (X key and Y rel) is-told and
    (X Y Z) delete and
    /
&

```

Эта программа содержит предложения **keys** (ключи), **fields** (поля), **zeroes** (нули) и **rels** (отношения). Отношение **addf** (добавить-поле) позволяет добавлять новые поля и при этом производит соответствующие изменения в списках полей, а также заносит нули в добавленные поля.

Для добавления и изменений ключей и отношений предназначено отношение **update** (перезаписать). Данное отношение обеспечивает необходимые изменения в записях базы данных, а также осуществляет замену старых предложений программы на новые.

Отношение **elim** (исключить) позволяет исключать предложения из любого отношения в программе пользователя. Отме-

тим, что при использовании данного отношения номер уничтожаемого предложения указывать не надо. В этом заключается его преимущество перед встроенным отношением микроПролога delete. Для более эффективного использования программы 5.6 ее необходимо оформить в виде модуля. Для этой цели к тексту программы следует добавить предложение

```
Module (update-mod (addf update elim
key field rel) (add delete is-told ON APPEND fields zeroes keys
rels))
```

Затем необходимо загрузить модуль MODULES и сохранить сформированный модуль в файле с именем UPDATE. Ниже приводится пример базы данных типа той, что предлагалось сгенерировать в упражнении 5.6, но которую теперь можно создать с помощью модуля BEGIN. Пусть в результате генерации была получена база данных следующего вида:

```
fields (sand cement bricks)
zeroes ( 0 0 0)
McDoo keys
in rels out
McDoo in ( 0 0 0)
McDoo out ( 0 0 0)
```

После окончания работы модуля BEGIN его следует удалить из памяти, выполнив команду KILL begin-mod, после чего ввести команду load UPDATE и приступить к заполнению базы данных, например, следующим образом:

```
all (: update x)
key X relation Y and new data Z ans McDoo in (14 6 700)
key X ... и т. д. ? ans McDoo out (5 2 120)
key X ... и т. д. ? ans Fee in (55 20 5000)
key X ... и т. д. ? just Fee out (17 6 600)
No (more) answers
```

Теперь, если вывести программу на экран, можно убедиться, что две старые записи с данными о хозяйстве фермера McDoo изменились, добавились записи для фермера Fee, а отношение keys теперь содержит список (McDoo Fee).

Следующий пример диалога показывает, как могут быть одновременно добавлены новый ключ и новое отношение:

```
all (: update x)
key X relation Y and new data Z? just Home stock (900 50 6000)
No (more) answers
```

Если теперь вывести программу на экран, можно обнаружить новое предложение для фермера Home, а предложения со

списками ключей (keys) и отношений (rels) выглядят следующим образом:

```
keys (McDoo Fee Home)
rels (in out stock)
```

Далее приведен пример добавления нового поля:

```
all (x : addf x)
X new field ? just blocks
blocks
No (more) answers
```

В результате списки в предложениях fields и zeroes будут выглядеть так:

```
fields (sand cement bricks blocks)
zeroes (Ø Ø Ø Ø)
```

Отметим, что для удаления предложения не нужно знать его положение в программе.

Упражнение 5.8

Попрактикуйтесь в применении модуля UPDATE к базам данных, построенным в упражнении 5.7.

5.7. ИСПОЛЬЗОВАНИЕ БАЗЫ ДАННЫХ

Нельзя предусмотреть заранее все пути доступа к базе данных, которые могут понадобиться пользователям. Однако, поскольку генерация базы данных с помощью модуля BEGIN осуществляется пользователем самостоятельно, он может обеспечить необходимые для него средства доступа.

Как уже было отмечено, режимы чтения и модификации данных часто предусматривают выполнение одних и тех же действий и иногда полностью совпадают. Рассмотрим в качестве примера базу данных для фирм, торгующих строительными материалами. Пользователю в ряде случаев удобно держать информацию о приобретенных и проданных материалах отдельно от сведений о действительном количестве этих материалов, находящихся в данный момент времени на складе. В таком случае с отношениями in и out можно использовать отношение stock, которое изменяло бы значения, соответствующие количеству различных материалов на складе. При другом варианте построения базы данных можно отказаться от раздельного хранения сведений о приобретении и продаже материалов. Вместо этого данные о торговых операциях можно суммировать с числами, характеризующими количество материалов на складе, но с разными знаками: с плюсом в случае приобретения и с минусом в случае продажи.

Ниже приводится программа пользователя, созданная с помощью модулей BEGIN и UPDATE. В данной программе 5.7 сведения о приобретении и продаже хранятся отдельно.

Программа 5.7

```
fields (sand cement bricks)
zeroes (0 0 0)
keys (McDoo Fee Home)
in rels out
McDoo in (60 15 500)
Fee in (200 55 1000)
Home in (500 120 3000)
McDoo out (20 4 250)
Fee out (90 10 550)
Home out (200 70 1900)
```

Программа 5.8 позволяет определить количество любого строительного материала, находящегося в настоящее время на складе.

Программа может, например, работать следующим образом:

```
all (x: McDoo stock x)
[определить все (x: McDoo имеет-на-складе x)]
(40 sand)
[40 песок]
(11 cement)
[11 цемент]
(250 bricks)
[250 кирпичи]
No (more) answers
[Ответов (больше) нет]
```

С помощью данной программы можно также получать количество какого-либо строительного материала у определенного торговца. Например, чтобы получить количество песка у торговца, необходимо выполнить следующую команду:

```
all (x: Fee stock (x sand)
[определить все (x : Fee имеет-на-складе (x песок))])
110
No (more) answers
[Ответов (больше) нет]
```

Упражнение 5.9

Напишите запросы для получения:

- 1) имен торговцев, на складе которых более 400 кирпичей (получить также число кирпичей у этих торговцев);
- 2) имен торговцев, имеющих песка больше, чем у Fee; определить также, сколько песка на складах у этих торговцев;
- 3) количество песка и цемента у каждого торговца.

Приведенные выше отношения match и stock должны быть присоединены к модулю UPDATE, служащему одновременно для чтения и модификации информации в базе данных.

Отношение match позволяет находить соответствующие друг другу элементы списков in и out для того, чтобы затем можно было вычислить количество определенного материала на данном складе. Указанное отношение может быть эффективно использовано, например, в том случае, когда в базе данных хранятся лишь итоговые величины. Последовательность действий при этом следующая: чтение данных, соответствующих введенным, затем вычисление их новых значений и, наконец, запись новых значений в базу данных. Для реализации указанных операций предназначены два отношения, описанные в программе 5.8.

Программа 5.8

```
change (X Y Z) if
  (key X rel Y change Z) is-told and
  changel (X Y Z)
changel (X Y Z) if
  (X Y x) delete and
  y isall (y: (z X1) match (x Z) and SUM (X1 z y)) and
  Y1 isall (Y1: Y1 ON y) and
  (X Y Y1) add and
```

Для практического использования отношений из программ 5.7 и 5.8 они должны быть оформлены в виде отдельного модуля с именем, отличающимся от других рассмотренных выше модулей. Новый модуль может быть назван, например, ACCESS.

Ниже приводится распечатка программы с базой данных, созданной с помощью модулей BEGIN и UPDATE. Эту базу данных можно использовать для проверки правильности работы модуля ACCESS. В ней содержатся сведения о количестве книг разного вида, предлагаемых несколькими книготорговыми фирмами.

```
fields (computers science gardening cookery thriller romance
western war)
zeroes (0 0 0 0 0 0 0 0)
keys (York Luton Leeds Bath Bury Hull)
in rels out
Luton in (100 70 120 85 70 200 90 50)
Leeds in (80 60 60 70 50 40 60 30)
Bath in (50 30 30 40 20 50 50 20)
Hull in (20 30 25 25 40 20 20 20)
York in (25 25 40 30 30 35 40 15)
Bury in (25 10 10 20 20 30 25 30)
York out (3 7 9 2 12 16 23 12)
Bath out (13 21 11 24 9 33 8 7)
Bury out (9 5 3 14 9 19 23 7)
Leeds out (32 30 27 45 20 19 39 18)
Hull out (6 4 9 8 2 3 7 4)
Luton out (35 22 68 41 27 104 52 18)
```

Для работы с базой данных достаточно уметь обращаться к ней при помощи отношений stock и change, что, как можно было заметить, не представляет труда даже для неподготовленного пользователя.

С помощью отношения change пользователь определяет, к какому отношению базы данных (in или out) принадлежит изменяемая запись, указывает соответствующий ей ключ и вводит данные, необходимые для модификации этой записи. Отношение changel осуществляет поиск в базе данных информации, соответствующей списку, введенному с помощью отношения change, а затем суммирует найденные и введенные данные, получая в результате исправленный список данных. Отношение isall используется в рассмотренном отношении changel дважды.

Итак, прежде чем приступить к практическому использованию приведенной выше базы данных со сведениями о книоторговых фирмах, необходимо оформить программу ACCENS в виде модуля, добавив к списку экспорта имена отношений stock и change и присоединив к списку экспорта модуля UPDATE имя отношения isall.

Ниже приводятся возможные варианты запросов к базе данных:

```
all (x y: x stock y)
Luton (65 computers)
Luton (48 science) ...
all (x y: x stock (y cookery))
Luton 44
Leeds 25 ...
all (x y z: x stock (y z) and y LESS 10)
Bath 9 science
York 3 war ...
```

В результате следующего диалога в базу данных будут внесены изменения:

```
all (x: change x)
key x rel y change x ? just Luton out (0 0 0 2 5 22 8 5)
(Luton out (0 0 0 2 5 22 8 5)
No (more) answers
```

В том, что изменения внесены правильно, можно убедиться с помощью следующего запроса, позволяющего вывести содержимое базы данных на экран:

```
all (x: Luton out x)
(35 22 68 43 32 126 60 23)
all (x: Luton stock x)
(65 computers)
(48 science)
(52 gardening)
(42 cookery) ...
```

Над информацией, хранящейся в рассматриваемой базе данных, можно выполнить операции сравнения. Приведенный ниже пример демонстрирует эту возможность:

all (x y: x stock (y war) and Leeds stock (z war) and y LESS z)
[определить все (x y: x имеет-на-складе (y книг-о-войне) и Leeds имеет-на-складе (z книг-о-войне) и y LESS z)]

YORK 3

No (more) answers

[Ответов (больше) нет]

В рассмотренных выше примерах, связанных с информацией о книготорговых фирмах, все изменения описывались положительными числами. Однако возможны случаи, когда изменения удобно представлять в виде отрицательных чисел. Такие ситуации могут возникнуть, когда фирме приходится принимать книги, возвращаемые по той или иной причине покупателем, или, в другом случае, самой возвращать книги другой фирме, у которой они были ранее приобретены. Приведенный ниже пример иллюстрирует второй из указанных случаев, который может возникнуть, когда, например, фирма (Bury) замечает спад покупательского интереса к романтической литературе.

all (: change x)

key X rel Y change Z ? just Bury in (0 0 0 0 0 - 25 0 0)

No (more) answers

Теперь можно убедиться в корректности внесенных изменений следующим образом:

all (x: Bury in x)

(25 10 10 20 20 5 25 30)

No (more) answers

Упражнение 5.10

Составьте запросы, позволяющие определить:

- 1) сколько книг-вестернов имеется у каждой фирмы;
- 2) названия фирм, у которых книг по садоводству больше, чем у фирмы York;
- 3) названия всех тех фирм, которые имеют больше книг по кулинарии, чем по садоводству;
- 4) общее количество художественной литературы у каждой фирмы.

Упражнение 5.11

Составьте запросы, позволяющие достаточно просто получить следующие сведения:

- 1) объемы поступлений и продаж по всем видам литературы, а также общее число книг на складе для некоторой фирмы;

2) число книг-вестернов, проданных данной фирмой;

3) прибыль, полученную данной фирмой в результате продажи определенного вида литературы при условии, что известны цены, по которым книги были куплены и проданы.

Выполнив предложенные выше упражнения, читатель, вероятно, смог убедиться в том, что сформировать запросы для получения сведений о поступлении на склад и продаже книг какой-либо одной тематики не вызывает особых затруднений. Однако сделать запрос о суммарных показателях по нескольким тематикам [см. упражнения 5.11, (1) и (3)] гораздо сложнее. Следует отметить, что именно запросы такого вида часто используются при обращениях к базам данных. В программе 5.9 приведены несколько дополнительных отношений, облегчающих составление запросов указанного типа. Эти отношения, а также предложения, позволяющие упростить определение тематик проданных и купленных книг, могут быть присоединены к модулю ACCESS.

Отношение tot первоначально определяет, что сумма элементов нужного списка равна нулю. Затем с помощью хвостовой рекурсии вычисляется сумма всех элементов списка. Следующие три отношения, использующие tot, служат для получения объемов поступлений, продаж и имеющейся в текущий момент на складе

Программа 5.9

```
( ) tot 0
(X Y) tot Z if
    Y tot x and
    SUM (X x Z)
X in-tot Y if
    X in Z and
    Z tot Y
X out-tot Y if
    X out Z and
    Z tot Y
X stock-tot Y if
    X in-tot Z and
    X out-tot x and
    SUM (Y x Z)
X in-type (Y Z) if
    X in x and
    fields y and
    (Y Z) match (x y)
X out-type (Y Z) if
    X out x and
    fields y and
    (Y Z) match (x y)
```

литературы. Запрос может быть, например, таким:

all (x y: x in z and z tot y)

С помощью отношений in-type и out-type можно для каждой фирмы получить объемы поступившей и проданной литературы определенной тематики. Например:

all (x y: Hull in-type (x y))
20 computers
30 science ...

Следует отметить, что рассмотренная выше программа может использоваться для работы с базой данных, содержащей сведения о приобретении и продаже каких-либо других предметов или материалов. Ниже приводятся несколько запросов, демонстрирующих возможности рассмотренных отношений.

1. Определить количество научной литературы на складе каждой фирмы.

all (x y: x out-type (y science))
York 7
Bath 21 ...

2. Найти фирмы, продавшие кулинарных книг больше, чем Bury.

all (x y: Bury out-type (z cookery) and x out-type (y cookery) and z
LESS y)
Bath 24
Leeds 45 ...

3. Найти фирмы, продавшие книг каждого типа меньше, чем это сделал Hull.

all (x y z: Hull out-type (X z) and x out-type (y z) and y LESS X)
York 3 computers
Bury 3 gardening ...

4. Найти фирмы, продавшие больше книг по садоводству, чем по кулинарии.

all (x y z: x out-type (y gardening) and x out-type (z cookery) and
z LESS y)
York 9 2
Hull 9 8 ...

Упражнение 5.12

Составьте запросы для получения:

- 1) названий фирм, имеющих на складе книг по вычислительной технике больше, чем York;
- 2) названий фирм, продавших книг каждой тематики больше, чем имеется таких же книг на складе фирмы Hull; найти и число проданных ими книг по соответствующим тематикам;
- 3) названий фирм, на складе которых книг любой тематики больше, чем книг по вычислительной технике, проданных фирмой Luton (найти также число и тематику книг);
- 4) названий фирм, продавших книг по любой тематике больше общего количества книг, имеющегося у фирмы Hull (определить также число и тематику книг).

Программа 5.10

```
X max if
    X ON Y and
    (forall Z ON Y and not Z EQ X then Z LESS X)
X min Y if
    X ON Y and
    (forall Z ON Y and not Z EQ X then X LESS Z)
X ave Y if
    Y tot Z and
    Y items x and
    TIMES (x X Z)
() items 0
(X Y) items Z if
    Y items x and
    SUM (x 1 Z)
frac (X Y Z) if
    TIMES (X Z Y)
percent (X Y Z) if
    frac (x Y Z) and
    TIMES (100 xX)
```

В программе 5.10 приведены другие полезные для работы с базой данных отношения: нахождение наибольшего и наименьшего элементов списка, среднего арифметического, а также относительной величины элемента списка, выраженной в процентах.

Ниже помещены примеры использования новых отношений.

1. Определить для каждой фирмы тематику книг, пользующихся наибольшим спросом:

```
all (x y z: x out X and y max X and x out-type (y z))
York 23 western
Bath 33 romance
```

2. Определить число полей в записях базы данных.

all (x: fields y and y items x)

8

No (more) answers

3. Определить для каждой фирмы тематику книг, объем продажи которых ниже среднего уровня продажи для данной фирмы.

all (x y z: x out X and Y ave X and x out-type (y z) and y LESS Y)

York 3 computers

York 7 science ...

Если пользователю приходится часто обращаться к базе данных с какими-либо однотипными, но сложными по форме запросами, то в этом случае к программе следует добавить отношение, позволяющее проще описывать запросы такого типа. Например, приведенное ниже отношение облегчает ввод запросов для получения суммарных объемов проданных книг по нескольким фирмам.

sales-tot X if

Y isall (Y: Z out-tot Y) and

Y tot X

Упражнение 5.13

Напишите отношение, облегчающее ввод запросов, позволяющих определить:

1) процентную долю проданных какой-либо фирмой книг по отношению к общему объему продажи;

2) общее число книг определенной тематики, проданных несколькими фирмами;

3) процентную долю общего количества книг одной тематики, проданных несколькими фирмами, по отношению к общему для этих фирм объему продажи книг;

4) тематику книг, пользующихся наибольшим спросом у покупателей;

5) тематику книг, пользующихся у покупателей наименьшим спросом.

Упражнение 5.14

Для обработки некоторых запросов упражнения 5.13 понадобится много времени. Опишите способы ускорения процесса получения ответа. Какими преимуществами и недостатками они обладают.

При выполнении последнего упражнения читатель должен был еще раз убедиться в том, что отношения, служащие для чтения и обновления базы данных, в значительной степени зависят от того, как база данных используется.

Можно значительно облегчить работу с базой данных, если включить в программу отношения, позволяющие упростить вид часто вводимых запросов. Так, например, для пользователей рассмотренной выше базы данных были бы удобны отношения, предназначенные для получения выражения в процентах показателей торговых операций и различных итоговых величин.

Расчет итоговых показателей часто требует больших затрат времени. В таких случаях целесообразнее хранить указанные показатели в базе данных, определив их значения заранее.

В некоторых прикладных задачах даже небольшие изменения необходимо вносить в базу данных немедленно. Такой режим работы с базой данных используется, например, в системе резервирования авиабилетов, когда в каждый момент времени нужны самые последние сведения о проданных и возвращенных билетах. В случае базы данных для книготорговых фирм такая оперативность совершенно не нужна. Сведения, хранящиеся в ней, могут понадобиться лишь при планировании закупок и продаж, производящемуся не столь уж часто.

Большая часть моделей микроЭВМ не может обеспечить ни достаточно высокой скорости обработки запросов, ни доступа к базе данных сразу нескольких пользователей. Поэтому для построения систем резервирования авиабилетов ЭВМ этого класса не годятся. Для микроЭВМ больше подходят задачи, где обновление информации производится с достаточно большими интервалами времени, например, ежедневно или раз в неделю, как в случае рассмотренной выше базы данных для книготорговых фирм. При таком режиме обновления информации вычисление и занесение в базу данных некоторых итоговых показателей реализовать будет несложно. Однако за ускорение процесса получения итоговых данных приходится расплачиваться увеличением объема памяти, занимаемой базой данных.

Ниже приводится полный текст программы, предназначенной для модификации и чтения записей базы данных книготорговых фирм (программа 5.11). В данной программе объединены отношения, имеющиеся в программах с 5.7 по 5.10, а также отношения, обеспечивающие более быстрое получение ответов на запросы, аналогичные тем, что нужно было получить в упражнениях 5.13 и 5.14. С помощью данной программы пользователь может изменять различные поля базы данных. После этого программа выполняет расчет различных итоговых показателей, таких, например, как общий объем книг, проданных каждой фирмой, общий объем продажи книг по каждой тематике, а также различных показателей, выраженных в процентах.

Поскольку указанные расчеты требуют довольно много вычислений, соответствующие процедуры будут выполняться долго.

Поэтому модификацию рассмотренной базы данных следует проводить через определенные промежутки времени и вводить сразу все накопленные к данному моменту исправления. Об окончании ввода пользователь сигнализирует, отвечая на запрос Пролог-системы сообщением just (последний ответ) или no (ответов больше нет).

Данная программа занимает довольно много места, поэтому, чтобы сэкономить память, следует ввести в Пролог-систему команду «KILL ergmes-mod» и удалить ненужные отношения модулей program-mod и query-mod так, как было описано выше.

Программа 5.11

```
(X Y) match ((X|Z) (Y|x)) .
(X Y) match ((Z|x) (y|z)) if
  (X Y) match (x z)
X stock (Y Z) if
  fields x and
  in (X y) and
  out (X z) and
  (Z X1) match (x y) and
  (Z Y1) match (x z) and
  SUM (Y1 Y X1)
change (X Y Z) if
  (key X rel Y change Z) is-told and
  changel (X Y Z)
change X if
  outtot KILL and
  intot KILL and
  salestot KILL and
  soldpercent KILL and
  typesales KILL and
  allsales KILL and
  seller KILL
change X if
  out (X Y) and
  Y tot Z and
  (X outtot Z) add and
  in (X x) and
  x tot y and
  (X intot y) add
change X if
  Y isall (Y: outtot (Z Y)) and
  Y tot X and
  (X salestot) add
change X if
  salestot (Y) and
```

```

    outtot (X Z) and
    percent (x Z Y) and
    (X soldpercent x) add
X change Y if
    X field and
    Y isall (Y: Z outtype (Y X)) and
    (X typesales Y) add
X change Y if
    typesales (X Z) and
    Z tot Y and
    (X allsales Y) add
(X Y) change (Z x) if
    y isall (y: allsales (X y)) and
    Z max y and
    x min y and
    allsales (X Z) and
    allsales (Y x) and
    (seller (X Z Y x)) add and
/
changel (X Y Z) if
    (X Y x) delete and
    y isall (y: (z X1) match (x Z) and SUM (X1 z y)) and
    Y1 isall (Y1: Y1 ON y) and
    (X Y Y1) add and
/
() tot 0
(X Y) tot Z if
    Y tot x and
    SUM (X x Z)
X max Y if
    X ON Y and
    (forall Z ON Y and not Z EQ X then Z LESS X) and
/
X min Y if
    X ON Y and
    (forall Z ON Y and not Z EQ X then X LESS Z) and
/
X ave Y if
    Y tot Z and
    Y items x and
    TIMES (x X Z) and
/
() items 0
(X Y) items Z if
    Y items x and
    SUM (x 1 Z) and
/
percent (X Y Z) if
    TIMES (x Z Y) and
    TIMES (100 x X) and
/

```

```

X stocktot Y if
    intot (X Z) and
    outtot (X x) and
    SUM (Y x Z)
X intype (Y Z) if
    in (X x) and
    fields y and
    (Y Z) match (x y)
X outtype (Y Z) if
    out (X x) and
    fields y and
    (Y Z) match (x y)
X field if
    fields Y and
    X ON Y
X key if
    keys Y and
    X ON Y

```

С помощью данной программы можно, например, обрабатывать информацию, хранящуюся в ранее рассмотренной базе данных для книготорговых фирм, содержащих отношения in, out, fields, keys и rels. Ниже приводится пример диалога пользователя с программой 5.11:

```

all (: change x)
[определить все (: изменения x)]
key X rel Y change Z ? ans Luton out (20 10 12 25 10 30 20 10)
[ключ X отношение Y изменение Z ? ответ Luton продал (20
10 15 25 10 30 20 10)]
key X rel Y change Z ? just Leeds out (5 5 5 10 8 6 5 3)
[ключ X отношение Y изменение Z ? последний-ответ Leeds
продал (5 5 5 10 8 6 5 3)].
...
..
No (more) answers
[Ответов (больше) нет]

```

Многоточиями здесь отмечены пустые строки, генерируемые программой после завершения очередного этапа процесса модификации. Если вместо использованного в приведенном примере запроса ввести «all (x : change x)», то вместо пустых строк будут выводиться изменяемые при модификации предложения. Несмотря на то, что описанный процесс выполняется долго, а база данных при этом значительно увеличивается в объеме, ответы на запросы типа тех, что требовались в упражнении 5.13, будут выдаваться практически без промедления, так как теперь после работы данной программы они оказываются заранее занесенными в базу данных.

Приведем краткое описание новых отношений, появившихся в последней рассмотренной программе:

outtot и intot	— общий объем закупок и продаж для фирмы
salestot	— общий объем продажи для нескольких фирм
soldpercent	— выражаемое в процентах отношение объема продаж одной фирмы к общему объему продаж
typesales	— количество проданных фирмой книг одной тематики
allsales	— количество книг одной тематики, проданных несколькими фирмами
seller	— число и тематика книг, пользующихся наибольшим и наименьшим спросом

Для того чтобы сохранить изменения, внесенные в базу данных в результате выполнения программы 5.11, программу с базой данных следует оформить в виде модуля, который после того, как модификация завершится, может быть убран из памяти командой KILL. Следует учитывать, что работа с отдельными модулями обеспечивает экономию памяти, а также повышает скорость последовательного доступа к базе данных.

С помощью рассматриваемой программы пользователь может выводить на экран информацию несколькими способами. Самый простой из них заключается в том, чтобы полностью выводить необходимые отношения. Отметим, что для такого способа пользователю не нужно знать структуру программы с базой данных, а в Прологе ему необходимы лишь минимальные сведения. Пользуясь отношениями описания программы, можно получать различные, в том числе и не совсем тривиальные сведения. Например, следующий запрос позволяет получить общий объем продажи книг по различным тематикам

```
list allsales
computers allsales 123
science allsales 164 ...
```

Чтобы найти процент от общего объема продаж для фирмы Bury, можно воспользоваться таким запросом:

```
&all (x: Bury soldpercent x)
7.9646852
No (more) answers
```

Кроме того, зная структуру программы, пользователь может извлекать из базы данных информацию, получение которой заранее не предусмотрено. Ниже приведен пример использования программы для определения тематики самой ходовой книги, объема полученной за нее прибыли и процентной оценки этой прибыли по отношению к общему объему прибыли.

all (x y z: seller (x y | X) and Y salestot and percent (z y Y))
 romance 230 2.0426287E1
 No (more) answers

Упражнение 5.15

С помощью рассмотренной программы получите следующую информацию:

- 1) число книг-вестернов, проданных каждой фирмой;
- 2) названия фирм, получивших прибыль менее 10% по отношению к общему объему прибыли;
- 3) названия фирм, продавших менее 100 книг;
- 4) общее число проданных книг по каждой тематике.

Упражнение 5.16

Дополните отношение change предложениями, которые позволяют получать:

- 1) процентную долю прибыли каждой фирмы по отношению к общей стоимости книг, поступивших в продажу;
- 2) наибольшее среди всех фирм число проданных книг одной тематики (одновременно выдавать также название фирмы и указанную тематику);
- 3) процентную долю прибыли некоторой фирмы, которая выручена за книги заданной тематики;
- 4) процентную долю прибыли некоторой фирмы за книги одной тематики по отношению к общему объему прибыли.

Ответы к упражнениям

Упражнение 5.1

- 1) all (x: Mntd form ((X | y) x))
- 2) all (x: Mntd form (y (x | z)))
- 3) all (x: Mntd form ((y | z) (X | Y)) and SUM (y X x))
- 4) all (x: Mntd form ((y z | X) (Y Z | x1)) and SUM (y Y x2) and SUM (z Z x3)) and TIMES (x2 3 x4) and SUM (x3 x4 x))
- 5) all (x y: Wsthd form ((x1 x2 x3 x4 x5) x6 and x form ((y1 y2 y3 y4) y5) and x4 LESS y)

Упражнение 5.2

См. упражнение 5.3

Упражнение 5.3

- 1) all (x: Mntd form ((x | y) z))
- 2) all (x y z: x htop y and x alltop z)
- 3) all (x y z: x atop y and x alltop z)

Упражнение 5.4

- 1) all (x y: x rec (for y) and 20 LESS y
- 2) all (x y: Evrtn hrec (for z) and x hrec (for y) and z LESS y
- 3) all (x y z: x rec (for z) and y rec (for z) and x LESS y
- 4) all (x y z: x rec (drew y) and x rec (lost z) and z LESS y
- 5) all (x y: Arsnl arec (won z) and x hrec (won y) and y LESS z

Упражнение 5.5

Возможен такой вариант базы данных:

held (sand cement bricks blocks gravel plaster paint)

key (McDoo Fee Putlog)

При этом удобно использовать отношение stock с записями следующего вида:

McDoo stock (0 0 0 0 0 0)

Упражнение 5.6

С помощью той же программы, что и в предыдущем упражнении, следует определить два отношения, например, так, как показано ниже:

X State relation name? ans stockin

X State relation name? just stockout

Отношение stockin предназначено для регистрации поступлений на склад, тогда как отношение stockout, наоборот, позволяет контролировать продажу материалов со склада.

Упражнение 5.7

Задания (1), (3) и (4) имеют аналогичные решения. Например, для задания (3) возможен следующий ответ:

all (: begin x)

X List all helds ? just (Computing Gardening Cooking Science Fiction)

X List all keys ? just (Alan Bill Colin Dora Eve Fred)

X List all relations ? just (owns)

Следует отметить, что отношения должны вводиться в виде списка даже в том случае, когда определяется одно отношение.

2. Возможны два варианта: список полей может содержать названия позиций на игровом поле (вратарь, левый защитник и т. д.) или номера игроков от 1 до 11.

5. Решение в общем аналогично предложенному в п. (3), но если ввод данных производится отдельными фрагментами, для некоторых улич могут быть введены несколько подписков типа (23 201 49A).

Упражнение 5.8

См. ответ к следующему упражнению.

Упражнение 5.9

- 1) all (x y: x stock (y bricks) and 400 LESS y)
- 2) all (x y: x stock (y sand) and Fee stock (z sand) and z LESS y)
- 3) all (x y z: x stock (y sand) and x stock (z cement))

Упражнение 5.10

- 1) all (x y: x stock (y western))
- 2) all (x y: x stock (y gardening) and York stock (z gardening) and y LESS z)
- 3) all (x: x stock (z gardening) and x stock (y cookery) and z LESS y)
- 4) all (X Y: X stock (x1 thriller) and X stock (x2 romance) and X stock (x3 western) and X stock (x4 war) and SUM (x1 x2 x5) and SUM (x3 x4 x6) and SUM (x5 x6 Y))

Упражнение 5.11

Для выполнения данного упражнения следует ознакомиться с описываемыми далее в тексте дополнительными отношениями.

- 1) all (x y: x out z and z tot y)
- 2) all (x y: z out z and holds X and (y western) match (z X))
- 3) если прибыль от сбыта книги-вестерна равняется 50 центам, запрос all (x y: x out z and fields X and Y western) match (z X) and TIMES (0.5 Y y)) позволит получить выражению в долларах прибыль за продажу книг-вестернов для каждой фирмы.

Упражнение 5.12

- 1) all (x y: x stock (y computers) and York stock (z computers) and z LESS y)
- 2) all (x y z: Hull stock (X z) and out-type (y z) and X LESS y)
- 3) all (x y z: Luton out-type (X computers) and x stock (y z) and y LESS X)
- 4) all (x y z: Hull out-tot X and x out-type (y z) and X LESS y)

Упражнение 5.13

- 1) X sold-percent Y if
sales-tot Z and
X out-tot x and
percent (Y x Z)
- 2) X type-sales Y if
X held and
Y isall (Y: Z out-type (Y X))
X all-sales Y if
X type-sales Z and
Z tot Y
S held if
holds Y and
X ON Y
- 3) X percent-sales Y if
sales-tot Z and
X all-sales x and
percent (Y x Z)
- 4) X bestseller Y if
Z isall (Z: x all-sales Z) and
Y max Z and
X all-sales Y and
- 5) X worstseller Y if
Z isall (Z: x all-sales Z) and
Y min Z and
X all-sales Y and

Упражнение 5.14

См. следующее упражнение.

Упражнение 5.15

- 1) all (x y: x outtype (y western))
- 2) all (x: x soldepercent y and y LESS 10)
- 3) all (x: x outtot y and y LESS 100)
- 4) либо list allsales, либо all (x y: x allsales y)

Упражнение 5.16

- 1) change X if
 X stocktot Y and
 X outtot Z and
 percent (x Z Y) and
 (X stockpercent x) add
- 2) X change (Y Z) if
 X out x and
 Y max x and
 helds y and
 (Y Z) match (x y) and
 (X outmax (Y Z)) add
- 3) change X if
 X outtype (Y Z) and
 Z allsales x and
 percent (y Y x) and
 (X typepercent (y Z)) add
- 4) change X if
 X outtype (Y Z) and
 x salestot and
 percent (y Y x) and
 (X allpercent (y Z)) add

Данные предложения добавляются к программе перед выполнением модуля UPDATE.

Разработка экспертных систем — это такая область искусственного интеллекта, которая в настоящее время привлекает всеобщее внимание. Более того многие исследователи считают понятия «искусственный интеллект» и «экспертные системы» синонимами. Автору кажется, что следует рассматривать все, что связано с созданием экспертных систем, как этап на пути к построению искусственного разума. По крайней мере, это разумный компромисс — ведь невозможно сконструировать устройство, которое смогло бы выполнить весь комплекс свойственных человеку интеллектуальных действий, но, с другой стороны, в различных достаточно узких прикладных областях могут быть созданы системы, которые обладают знаниями и умеют оперировать ими практически так же, как и человек, являющийся экспертом в этой прикладной области. Аналогично многим исследователям, экспертные системы практически бесполезны за пределами конкретной прикладной области. Так, например, экспертные системы, обладающие значительными знаниями в области разведки нефтерожждений, окажутся бесполезными, если речь пойдет о футболе.

Можно ожидать, что экспертные системы будут выполнять следующие функции:

- хранить специализированные знания и выдавать их по запросам;

- осуществлять логический дедуктивный вывод с использованием «нечеткой» логики;

- обладать способностями к обучению;

- уметь принимать решения или предоставлять информацию для принятия решений в области финансовой деятельности, геологической разведки, построения и тестирования разного рода систем и во многих других областях;

- объяснять свое поведение.

Пятое поколение — это термин, используемый в Японии для обозначения перспективных ЭВМ и новых принципов построения экспертных систем, которые впоследствии планируется использовать как базовые для разработки программного обеспечения. Правда, часто забывают, что реализация проекта пятого поколения ~~разачтывается~~ ^{начнется}, по крайней мере, на десять лет. Аналогич-

ные попытки делаются и в европейских странах. Так, финансовая поддержка была оказана исследователям по разработке основанных на знаниях интеллектуальных систем.

Примером сотрудничества в этой области является объединение усилий двух фирм Rascal и Norsk Data. Они разрабатывают мощные 32-разрядные системы, ориентированные на обработку знаний. Эти системы базируются на процессорах ND-500 и ND-100, имеют разделяемую оперативную память, предназначенную для параллельной работы нескольких пользователей, снабжены графическими терминалами с исключительно высокой разрешающей способностью (1200×1200) и с независимыми окнами, доступными с помощью клавиатуры или «мыши». Базовым языком для этих ЭВМ служит Zeta Лисп, созданный в МТИ, кроме того, в некоторых приложениях предусматривается использование Пролога. Эти системы планируется использовать главным образом в университетах, но часть из них должна найти применение в химических компаниях, в частности для проведения разведки нефте-рождений.

На первый взгляд, создать экспертную систему очень просто. Для этого надо найти эксперта, заставить сообщить все его знания в данной предметной области, ввести эти знания в ЭВМ, добавить в ЭВМ несколько правил по использованию знаний — и можно смело использовать вместо эксперта ЭВМ. В действительности процесс оказывается более сложным. Ведь, отнюдь, не очевидно, что эксперт согласится участвовать в предприятии, которое в конце концов может привести к тому, что он, отдав свое главное достояние — знания, — окажется ненужным. Даже если считать, что этот барьер будет преодолен, останется еще много факторов, которые сложно выразить словами, но которые оказывают существенное влияние на исход экспертизы. Многие из этих факторов определяются почти интуитивно, и эксперт зачастую не может передать их системе, поскольку и сам точно не знает, как ими пользоваться.

Другие трудности связаны с тем, что эксперт обычно не является специалистом по программированию, а программист может ничего не знать о довольно тонких моментах, характерных для данной предметной области. Эти трудности можно преодолеть с помощью нового типа специалиста — инженера по знаниям. Инженерия знаний как научная дисциплина определяет, каким образом получать знания от эксперта, как разрабатывать системы по обработке знаний и проектировать программы, ориентированные на применение этих знаний. Важным аспектом инженерии знаний является разработка самих систем инженерами знаний. Почти наверняка для экспертных систем потребуется быстродействующая память большого объема и не универсальная, а специализированная ЭВМ. Вероятно, экспертные системы будут поставляться в виде оболочек, которые дадут возможность пользователям создавать на их основе собственные системы, а не в форме

законченных пакетов, ориентированных на конкретные приложения.

Можно предположить, что экспертные системы легко адаптировать к решению самых разнообразных задач диагностики. Например, в области медицины экспертные системы могут быть использованы пациентами в случае неопасных заболеваний для самодиагностики и врачами для определения диагноза более серьезных болезней. Экспертные системы могут оказать помощь инженерам по электронике и механике в обнаружении и устранении неисправностей в работе сложной аппаратуры.

В самых разнообразных областях в проектировании электронных и механических устройств, в разработке компьютеров и языков программирования, в планировании экономических процессов экспертные системы можно использовать либо для создания законченных систем, либо как инструмент для проектирования и разработки отдельных подсистем. С помощью экспертных систем можно достаточно быстро создавать прототипы компьютерных систем и языков программирования. Дело в том, что экспертные системы предоставляют средства для создания тех основных элементов, которые являются общими для всех систем такого рода, а эксперты производят окончательный отбор и оснащение системы необходимыми для работы в конкретной области деталями. В этой и других областях знания эксперта используются людьми, которые либо не являются специалистами в этой области, либо разбираются в существе дела, но значительно хуже, чем ведущие эксперты.

Ниже представлена простая экспертная система, которая позволяет пользователю осуществлять контроль за работой магнитофона. Программа 6.1, реализующая экспертную систему, не содержит специальных знаний, и будет понятна практически всем. Проанализировав работу этой программы, пользователи смогут создавать значительно более сложные системы.

Программа 6.1

```
(X Y) match ((X Z) (Y x))
(X Y) match ((Z x) (y z)) if
  (X Y) match (x z)
check X if
  Y fault and
  Y checks Z and
  x ON Z and
  not (x) is-told and
  Y action y and
  (x X) match (Z y) and
  /
X fault if
  (fault X) is-told and
  faults Y and
  X ON Y
```

faults (no-sound no-play amplifier supply motor)
no-sound checks (plugged-in switched-on volume-up mains-fuse mains-lead sound-ok)
no-play checks (cassette-in cassette-rewound on-play tape-moving playing)
amplifier checks (speaker-leads speaker (Vcc 12v) T1-ok amp-ok)
supply checks (DC-fuse xformer-leads xformer rectifier-leads rectifier C1-ok (12v ok))
motor checks ((12v on motor) motor-running drivebelt idler pinchwheel pressure-pads all-ok)
no-sound action (plug-in switch-on turn-vol-up replace-fuse repair-lead (try supply fault))
no-play action (insert-cassette rewind switch-to-play (try motor fault) (try amplifier fault))
amplifier action (repair-leads new-speaker (try supply fault) replace-T1 (try head fault))
supply action (replace-fuse repair-leads new-xformer repair-leads new-rectifier replace-C1 (try amplifier fault))
motor action ((try supply fault) new-motor new-belt new-idler new-pinchwheel new-pads (try amplifier fault))

Укажем для неспециалистов, что в программе 6.1 используются следующие обозначения:

xformer — трансформатор;
 С — конденсатор;
 Т — транзистор;
 leads — провода или соединения на печатных платах.

Отношение faults включает все возможные типы неисправностей, в то время как отношение checks описывает различные процедуры контроля (тестирования), применяющиеся для установления причин возникновения этих неисправностей. Пользователю после того, как он информирует систему о неисправности, будет рекомендовано применить одну из этих процедур. Кроме того, процедуре тестирования соответствует специальное действие, которое пользователь должен выполнить для устранения неисправности.

Соответствующие друг другу неисправности, процедуры тестирования и действия по устранению неисправности связываются с помощью отношения match, которое было использовано ранее для выбора упорядоченных пар из двух списков. В некоторых случаях действия по устранению неисправности заменяются указанием ввести новый тип неисправности для того, чтобы получить больше информации о причине неисправности. Это свидетельствует о том, что система исчерпала свои знания о первоначально введенной неисправности.

В списках процедур контроля сначала идут элементарные процедуры, такие, как проверка, вставлена ли вилка в штепсель и включено ли устройство, но в дальнейшем процедуры

становятся более технически сложными. Для того чтобы начать работать с программой, пользователю необходимо использовать в запросе отношение check. После этого система попросит его ввести тип неисправности. Затем необходимо будет провести ряд предлагаемых системой процедур тестирования и сообщить системе их результат в виде ответа yes/no. В случае положительного ответа система считает, что все в порядке и не рекомендует осуществлять никаких действий; в случае отрицательного ответа пользователю сообщается, что следует предпринять. Если ни одна из связанных с неисправностью процедур тестирования не позволяет выявить причину неисправности, пользователю обычно предлагается выполнить еще один ряд процедур тестирования. Покажем, как может быть обнаружена причина простейшей неисправности:

```
all (x: check x)
[определить все (x: проверка x)]
fault X ? ans no-sound
[неисправность x ? ответ нет-звука]
plugged-in ? y
[вставлена-ли-вилка-в-штепсель ? да]
switched-on ? y
[включено-ли-устройство ? да]
volume-up ? no
[включен-ли-звук ? нет]
turn-vol-up
[включить-звук]
No (more) answers
[Ответов (больше) нет]
```

Следующий протокол работы программы иллюстрирует тот случай, когда основные процедуры тестирования не позволяют обнаружить причину неисправности и пользователям рекомендуется использовать ряд процедур, относящихся к другому типу неисправности:

```
all (x: check x)
[определить все (x: проверка x)]
fault X ? ans no-sound
[неисправность X ? ответ нет-звука]
plugged-in ? y
[вставлена-ли-вилка-в-штепсель ? да]
switched-on ? y
[включено-ли-устройство ? да]
volume-up ? y
[включен-ли-звук ? да]
mains-fuse ? y
[оболочка-фидера-в-порядке ? да]
mains-lead ? y
```

```

[провода-фидера-в-порядке ? да]
sound-ok ? по
[звук-в-порядке ? нет]
(try supply fault)
[(проверьте работу источника питания)]
No (more) answers
[Ответов (больше) нет]

```

После этого пользователь может продолжать работу с системой таким образом:

```

all (x: check x)
[определить все (x: проверка x)]
fault X ? ans supply
[неисправность X ? ответ источник-питания]
DC-fuse ? у
[предохранитель-постоянного-тока-в-порядке? да]
xformer-leads ? у
[провода-трансформатора-в-порядке ? да]
xformer ? у
[трансформатор-в-порядке ? да]
rectifier-leads ? у
[соединения-выпрямителя-в-порядке ? да]
rectifier ? по
[выпрямитель-в-порядке ? нет]
new-rectifier
[заменить-выпрямитель]
No (more) answers
[Ответов (больше) нет]

```

Элементарные процедуры тестирования рассчитаны на достаточно поверхностно знакомых с предметной областью пользователей, а не на умудренных опытом специалистов. Естественно, что специалисты хотели бы начинать работать с системой только после того, как простейшие процедуры тестирования не дали результата. Им хотелось бы вводить в систему знакомые понятия, такие, как источник питания, усилитель и т. п., и получать в ответ данные о причине неисправности, относящиеся к конкретным элементам устройства. Приведенный ниже пример дает возможность установить, что неисправна емкость C1 источника питания:

```

all (x: check x)
[определить все (x: проверка x)]
fault X ? ans supply
[неисправность X ? ответ источник питания]
DC-fuse ? у
[предохранитель в порядке ? да]
xformer-leads ? у
[провода трансформатора в порядке ? да]
xformer ? у

```

[трансформатор в порядке ? да]
 rectifier-leads ? у
 [соединения выпрямителя в порядке ? да]
 rectifier ? у
 [выпрямитель в порядке ? да]
 C1-ok ? по
 [емкость C1 в порядке ? нет]
 replace C1
 [заменить C1]
 No (more) answers
 [Ответов (больше) нет]

Напомним, что положительный ответ означает, что тестирование данного элемента проведено и никаких отклонений в его поведении не обнаружено. Вопрос системы C1-ok ? предполагает, что специалисты либо сразу на основании данных уже проведенных процедур тестирования готовы дать ответ о работе емкости C1, либо им потребуется какой-нибудь специальный тест для C1. В принципе нет достаточных оснований для того, чтобы не давать более полной информации пользователям-неспециалистам. Например, для каждого элемента может быть предусмотрено специальное отношение action, описывающее процедуры тестирования, позволяющие установить, является ли данный элемент причиной неудовлетворительной работы устройства.

Упражнение 6.1

Покажите, как можно использовать экспертную систему для обнаружения неисправности маховика механизма воспроизведения. Причем учтите, что неспециалисты (а) и специалисты (б) будут взаимодействовать с системой по-разному.

Ниже показано, как можно расширить систему, обеспечив пользователей более детальной информацией для поиска неисправностей

Программа 6.2

faults (no-sound no-play amplifier supply motor transistor gain)
 amplifier action (repair-leads new-speaker (try supply fault) (try
 transistor fault) (try head fault))
 transistor checks (tested-junctions gain-ok)
 gain checks (tested-gain amplifier-ok)
 transistor action ((If NPF type connect ohmmeter pos to b : resistance
 to c and to e should be low. Reserve leads: resistance should be high.
 Reverse polarities for PNP type.) (Try gain fault))
 gain action ((if NPN type connect ohmmeter between c and e:
 resistance should be high. Connect 4K7 between c and b: resistance
 should fall. Reverse polarities for PNP type) (Try amp-circuit fault))

Выше вопрос

T1 ok ?

предусматривал, что пользователь обладает знаниями о транзисторах и может контролировать их работу. Если же таких знаний пользователь не имеет, то система должна быть готова предоставить их ему. Слова transistor (транзистор) и gain (коэффициент усиления) должны быть добавлены к списку неисправностей, а слова try transistor fault (проверьте работу транзистора) должны появиться в предложении о действиях в случае неисправности усилителя вместо слов replace T1 (заменить T1).

После элементарных типов неисправностей в списке стоят более технически сложные дефекты. Им соответствуют специальные процедуры тестирования, ориентированные на проверку конкретных элементов устройства. Поэтому для того чтобы понимать смысл выполняемых операций, необходимы знания в области электроники.

Экспертные системы, подобные только что описанной, могут использоваться в качестве обучающих пакетов, а также для поиска неисправностей в различных устройствах. Информация о том, как проводить тестирование транзисторов, специалистам не нужна, им необходимо только знать, какой именно транзистор или группа транзисторов являются причиной неисправности. Начинаящим электронщикам информация о том, как тестировать транзистор, понадобится только первые несколько раз; в дальнейшем никаких подсказок им не потребуется.

Еще раз отметим, что систему можно использовать либо как пакет для обучения основам обслуживания электронных устройств, либо как советчик для опытных специалистов, которые эксплуатируют незнакомое оборудование. В каждом из этих двух случаев, после того как люди приобретут необходимые навыки, система уже не понадобится.

Возможности системы могут быть расширены с помощью введения в нее знаний о новых неисправностях и необходимых для их устранения действиях. Эти знания передаются системе экспертами. В результате знания экспертов становятся доступными менее квалифицированным специалистам. Ниже приведено несколько отношений, использование которых позволяет системе получить информацию о новых неисправностях, процедурах тестирования и действиях по устранению этих неисправностей.

Программа 6.3

```
new-fault X if
(X new-fault) is-told and
faults Y and
not X ON Y and
(faults Y) delete and
APPEND (Y (X) Z) and
(faults Z) add and
new-checks X and
```

```

new-checks X if
(Y what tests are needed) is-told and
(X checks Y) add and
new-action X and
/
new-action X if
(Y what action is taken) is-told and
(X action Y) add and
/

```

Информация о процедурах тестирования и необходимых действиях по устранению неисправностей вводится в форме списков. Каждый элемент списка представляет собой либо два связанных дефисом слова, либо последовательность слов, заключенную в скобки. Использовать и дефис, и скобки для определения одного элемента списка запрещается. Покажем теперь, как можно ввести в систему информацию о новой неисправности-искажении (distortion):

```

all (x: new-fault x)
[определить все (x: новая-неисправность x)]
X new fault ? ans distortion
[X новая неисправность ? ответ искажение]
X what tests are needed ? ans (motor-speed (Vcc = 12v) motor
(replay head tight) ( replay head aligned) (replay head ok) ampli-
fier-bias (speaker cone not torn))
[X какие процедуры тестирования нужны ? ответ (скорость-
электродвигателя (напряжение = 12 В) электродвигатель
(плотно прилегает головка воспроизведения) (выравниена голов-
ка воспроизведения) (головка воспроизведения в порядке)
напряжение смещения усилителя (не прорван диффузор громко-
говорителя))]
X what action is taken ? ans (adjust-VR1 (try supplu fault)
replace-motor tighten-head align-head replace-head adjust-VR5
replace-speaker
[X какие действия по устранению неисправностей необходимо
предпринять? ответ (отрегулировать-VR1 (проверить источник
питания) заменить-электродвигатель прижать-головку выров-
нять-головку заменить-головку отрегулировать-VR5 заменить-
громкоговоритель)]

```

Если теперь вывести на экран дисплея текст программы, то можно будет убедиться в том, что, во-первых, в списке неисправностей появилось слово distortion (искажение) и, во-вторых, в программу добавлено два новых предложения для определения причины искажений и их устранения. Если возникнет необходимость, то с помощью описанной выше процедуры можно построить законченную систему.

Но все-таки первоначально в программу должны быть включены отношения `match`, `check`, `fault`, `new-fault`, `new-checks`, `new-action` и, по крайней мере, один элемент должен присутствовать в списке отношения `faults`. Если все это есть, то пользователи могут приступать к введению в систему новых типов неисправностей, относящихся к ним процедур тестирования и действий по их устранению. Таким образом, программа представляет собой оболочку, использование которой дает возможность пользователям создавать экспертные системы, ориентированные на конкретную предметную область.

Имена отношений `faults` и `checks` удобны для систем, предназначенных для поиска неисправностей, но, естественно, эти имена могут быть изменены по желанию пользователей. Независимо от выбранных имен отношений основной принцип построения экспертных систем остается неизменным.

Упражнение 6.2

Используйте приведенные выше отношения для проектирования экспертной системы, предназначенной для поиска неисправностей, либо в любой известной Вам технической системе — например, такой, как центральное отопление, освещение, либо в устройстве типа радиоприемника, телевизора, автомобиля и т. п.

Упражнение 6.3

Измените оболочку таким образом, чтобы она оказалась пригодной для генерации экспертных систем, не связанных с поиском неисправностей. Попробуйте построить экспертную систему для идентификации объектов, относящихся к различным типам, например, для произведений живописи (учитывая данные о стиле, используемых материалах и т. п.), растений (здесь можно принять во внимание высоту, цвет, характер произрастания и т. п.) или для предметов любой другой хорошо знакомой Вам области.

6.1. СПЕЦИАЛИЗИРОВАННАЯ СИСТЕМА

Многие операции, выполняемые в процессе серийного производства продукции, аналогичны операциям, осуществляемым при поиске неисправностей оборудования. Так, например, для контроля за процессом производства и для обеспечения качества продукции обычно используются заранее подготовленные тестовые процедуры. Во многих случаях процесс производства целесообразно разбить на отдельные шаги и представлять в виде специальной схемы. Эту схему в дальнейшем можно использовать совместно со справочником, содержащим более подробно сведения о характере производства. Вся схема полностью должна находиться в

распоряжении всех подразделений, занятых в производстве. Кроме того, каждое подразделение должно располагать исчерпывающими знаниями о той части производства, в которой она занята.

На рис. 6.1 приведена часть такой схемы для описания в предыдущем разделе экспертной системы. Те части схемы, которые отсылают к процедурам, проверяющим, вставлена ли вилка в штепсель, включено ли устройство и т. п., содержат почти столько же информации, сколько и программа. Но в части схемы, относящейся к последующим более сложным процедурам, указываются только названия, характеризующие тип выполняемой процедуры тестирования, и связь ее с другими элементами схемы.

Экспертная система может быть ориентирована не только на поиск причин неисправностей, но также и на создание и выдачу в удобной пользователю форме схемы всего процесса поиска. Это позволяет пользователю в случае необходимости пропускать часть процедур контроля и сразу переходить к той последовательности действий, которая, с его точки зрения, быстрее приведет к успеху.

На рис. 6.2 приведена часть более сложной схемы, на которой изображены различные этапы производства печатных плат. Совершенно не обязательно знать в деталях, как выполняются разные операции в процессе производства высококачественных печатных плат и насколько точными являются используемые типы и стандарты. Не беспокойтесь, если некоторые из процедур, упомянутых на рис. 6.2, окажутся Вам неизвестными. Предполагается разработать две независимые экспертные системы для производства печатных плат. Первая, обучающая, предназначена для описания процессов обработки, используемых в производстве; вторая будет представлять собой указатель к справочнику о производстве печатных плат и, кроме того, с ее помощью можно получить информацию о том, какие процессы выполняются на разных этапах производства.

В принципе может быть создана и третья экспертная система, в состав которой будет входить вся информация из справочника,

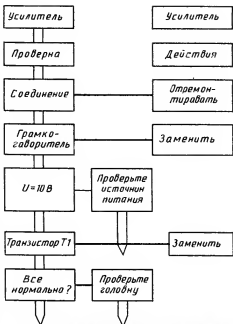


Рис. 6.1. Схема поиска неисправностей

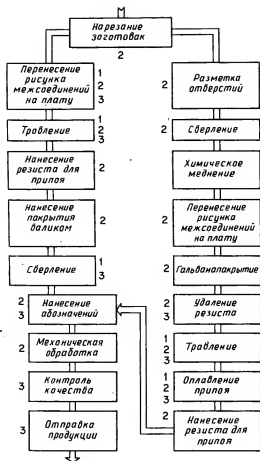


Рис. 6.2. Схема процесса производства печатных плат (цифрами обозначены разные виды контроля:

1 — контроль первого образца;
2 — операторский контроль;
3 — обеспечение качества)

но дело в том, что персональная микроЭВМ не обладает оперативной памятью достаточного объема для работы с программами такого размера. Все-таки некоторые рассматриваемые отношения дадут пользователям представление об экспертных системах большого размера.

Читатель, незнакомый с процессом производства печатных плат, после разбора следующего далее материала должен быть в состоянии построить экспертную систему для хорошо известной ему предметной области.

Обучающая экспертная система состоит из двух независимых программ: первая (программа 6.4) содержит данные об общих положениях производства печатных плат, вторая (программа 6.5) включает более детальную информацию. Текст первой программы следует ниже.

Программа 6.4

explain X if

(Y Do you want general or detailed information) is-told and

Y info X

general info X if

(Y Select Design QA (Quality Assurance) Flow-chart Route-card)

is-told and Y data X

detailed info (Kill this program and load PCB2) if

(Finished with this program) is-told and

Design data (Design involves the preparation of artwork several times the final board size on drafting film. Computer Aided Design may be used to produce digitized layout of tracks, holes and pads, which is stored on tape or disk.)

QA data (Quality Assurance guarantees that boards are made to correct standards. PCBs do not pass a stage of manufacture until samples are taken and tests performed. Samples and test results are stored and must be available to the Standards Authority on demand))

Flow-chart data (The flow chart details each stage in the manufacturing process and the QA tests which are needed to meet the relevant standards))

Route-card data (The route-card contains similar information to the flow-chart and accompanies the job through all processing stages to show which processes are complete and which remain to be carried out.))

Программа является самодокументирующей. Она довольно коротка, но при желании в отношении data могут быть включены дополнительные утверждения, содержащие гораздо больше информации. Если информации, относящейся к одному объекту, так много, что она не помещается на листе, то можно использовать два предложения и более с идентичными заголовками.

Аналогичная организация отношений может быть, конечно, использована и в любой другой предметной области. Ниже приведена программа 6.5, имеющая в общем такую же организацию и те же самые имена отношений, что и предыдущая программа. Но в этой программе больше предложений, содержащих подробную информацию о различных аспектах процесса производства печатных плат.

Программа 6.5

explain X if

(Y do you want information on PTH of conventional boards)

is-told and

7 CLS and

Y info X

PTH info X if

(Select machining electroless electroplate resist primary-image strip-resist strip-Snpb gold-plate etch reflow legend-print QA) is-told and

Z CLS and
 Y data X
 conventional info X if
 (Y select from general image roller-coat drilling) is told and
 Z CLS and
 Y data X
 machining data (Machining includes drilling, sawing, routing,
 punching and profile cutting on guillotines. Tooling holes are
 punched or drilled for accurate registration.)
 electroless data (Chemicals are used to deposit a thin film of copper on
 the boards and in the plated-through holes to make them conductive
 for subsequent plating.)
 electroplating data (Copper is deposited by electroplating the entire
 surface of the board. This is followed by tin-lead (SnPb) plating of the
 eventual conducting areas. Typical depth: Copper 20, SnPb 10
 microns.)
 resist data (Resists are materials or solutions which protect parts of
 the board from some processes. SnPb resists the acids used for
 etching. Solder resist is applied by silk screen or dry film to parts of
 the board which will not be soldered.)
 primary-image data (The conductor pattern is formed by laminating
 dry film onto the board followed by exposure to UV. Silk screen
 printing with acid-proof inks is also used.)
 strip-resist data (The dry film is stripped from those areas where
 copper is to be etched away. Those areas will have been softened by
 exposure to UV.)
 strip-SnPb data (A selective chemical stripper is used to remove
 off connects those areas together.)
 masked for this purpose.)
 gold-plate data (Gold is plated to a depth of 2.5–5 microns on areas
 such as edge connectors. A plating bar which is later cut
 off connects those areas together.)
 etch data (The copper is etched from areas not covered by resist in
 solutions such as ferric-chloride or cupric-chloride.)
 reflow data (Tin-lead surfaces are brought to the melting point in hot
 oil, vapour or by means of infra-red to enable them to be soldered.)
 legend-print data (Code numbers, circuit symbols etc. are silk-screen
 printed in solvent and solder-resistant inks.)
 QA data (All the previous operations are subject to strict quality
 control. Sampling and testing is followed by QA release and a
 Certificate of Conformity to the agreed standard.)
 general data (Conventional boards are those which do not have plated
 through holes (PTH). Tracks on double-sided boards are connected
 through to the second side by means of studs or soldered links.)
 image data (Conventional (print and etch) boards are not drilled until
 after the primary imaging stage. Conductor patterns are formed by
 laminating dry film, exposure to UV and developing. A stencil of the
 pattern is made from original photographs.)
 roller-coat data (Flux covered boards pass through a roller-coating
 machine which covers exposed surfaces with SnPb to a depth of
 1–2 microns.)
 drilling data (For hole sizes of 0.8 mm and above, stacks of up to 5
 boards are pinned together and drilled by sight, template or NC
 machine. Smaller holes are drilled singly.)

Приведенная выше программа довольно велика для версии микроПролог ЭВМ Spectrum. Поэтому желательно не загружать модуль TOLD до тех пор, пока вся программа не введена, а после ввода и перед выполнением убрать модуль PROGRAM. Типичный протокол работы программы дается ниже:

all (x: explain x)

[определить все (x: объяснить x)]

X Do you want information on PTH or conventional boards ?

ans PTH

[X Вы хотите получить информацию о методе, использующем сквозные металлизированные отверстия, или о стандартном методе? Ответ метод, использующий стандартные металлизированные отверстия]

X Select form machining electroless electroplate resist primary-image strip-resist gold-plate etch reflow legend-print QA ? ans legend-print

[X Выберите один из этапов: механическая-обработка химическая-активация гальванопокрытие нанесение-резиста перенесение рисунка межсоединений на плату снятие-резиста нанесение-на-плату-условных-обозначений обеспечение-качества ? ответ нанесение-на-плату-условных-обозначений]

Code-number, circuit symbols...

[Кодовые числа, символические обозначения схем...]

No (more) answers

[Ответов (больше) нет].

Терм 7 CLS используется в трех первых предложениях программы 6.5 для того, чтобы очистить экран дисплея и таким образом подготовить его для вывода новой информации. Цифра 7 позволит установить белый цвет экрана; пользователь, естественно, может выбрать любой другой цвет, воспользовавшись для этого соответствующим числом.

Упражнение 6.4

Используйте программу 6.5 для получения информации;
а) о травлении; б) о нанесении покрытия с помощью валика.

6.2. СПОСОБНОСТЬ К ОБУЧЕНИЮ

Мощь экспертной системы значительно возрастет, если в процессе работы она будет обладать способностью к обучению, т. е. к усвоению новых фактов и методов их обработки. Достичь этого можно различными способами. Так, система может проводить статистическую обработку данных и формировать новые правила из таким образом получаемой информации. Кроме того, систему можно связать с реальным миром с помощью специальных сенсор-

ных устройств, которые позволяют ей собирать данные об этом мире и на этой основе формировать правила. И, наконец, система может приобретать знания в результате опроса экспертов. В программе 6.6 показано, как нужно модифицировать предыдущую программу (6.5), чтобы она могла приобретать новые знания в процессе общения с экспертами.

Программа 6.6

```

add-date X if
    add-heading Y and
    X ON Y and
    (X data Z) is-told and
    (X data Z) add
add-topic X if
    (topic Y) delete and
    (new topic X) is-told and
    APPEND (Y (X) Z) and
    (topic Z) add and
/
add-heading X if
    add-topic Y and
    (headings (X) for topic Y) is-told and
    (Y heading (X)) add and
/
explain X if
    topic Y and
    (Z Select a topic from: Y) is-told and
    Z info X
X info Y if
    X heading Z and
    (x Select heading from: Z) is-told and
    7 CLS and
    x data Y
topic (PTH conventional)
PTH heading (machining electroless electroplate resist primary-image
strip-resist gold-plate etch reflow legend-print QA)
conventional heading (general image roller-coat drilling)
machining data (Machining includes . . . ) etc.

```

Во время обучения нет необходимости дополнять все предложения data. Например, первое предложение (о машинной обработке), относящееся к методу, использующему сквозные металлизированные отверстия и первое предложение (об общих принципах), относящиеся к стандартному методу обработки, могут быть составлены без изменения. В новом варианте программы пользователю, работающему с отношениями explain и info, предстоит осуществлять выбор не из фиксированного списка, а из списка с переменным числом членов. Причем длина и содержимое этого списка определяются информацией, получаемой от эксперта.

Таким образом, программа предназначена для выполнения двух функций: во-первых, она является оболочкой, пользуясь которой

эксперты могут создавать экспертные системы, и, во-вторых, она дает возможность работать с готовыми экспертными системами. Имена основных категорий рассматриваемой предметной области хранятся в отношении topic, а понятия, описывающие категории, — в отношении heading. И, наконец, подробная информация по каждому из понятий содержится в различных предложениях отношения data. Все только что упомянутые отношения — topic, heading и data — могут быть дополнены в процессе диалога с экспертом. Чтобы не усложнять программу, в нее не включены средства для модификации и удаления предложений отношения data; в случае необходимости такие средства могут быть легко введены.

Предположим теперь, что необходимо добавить в программу информацию о новом методе производства печатных плат — о поверхностном монтаже. Покажем, как это может быть сделано:

```
all (x: add-data x)
[определить все (x: добавить-данные x)]
new topic X ? ans SMD
[новая категория X ? ответ SMD]
headings (X) for SMD ? just (meaning methods advantages)
[понятия (X) для SMD ? последний ответ (смысл методы достоинства)]
meaning data X ? just (Surface mounting of devices (SMD) is
a method
[смысл данные X ? последний ответ (поверхностный монтаж
устройств (SMD) — это метод]
of fitting ICS and other components of PCB
[сборки интегральных схем и других компонентов печатных
плат, осуществляемый без]
without the need for drilling holes for the leads.)
[использования отверстий для соединений])
meaning
[смысл]
methods data X ? just (Components are temporarily held in place
[методы данные ? последний ответ (компоненты временно удерживаются на нужных местах]
with adhesive and then soldered by means of hot vapours
[клеяким составом, а затем припаиваются с помощью горячих
испарений]
which melt solder cream smeared around the joints.)
[которые расплавляют припой, нанесенный на места соединений])
methods
[методы]
advantages data X ? just (SMDs provide higher density circuits
[достоинства данные X ? последний ответ (метод поверхностного
монтажа обеспечивает более высокую плотность размещения
элементов]
are more reliable, require no drilling and
```

[является более надежным, не требует отверстий для соединений и]
are cheaper to fit than other types.)
[дешевле аналогичных методов)]
advantages
[достоинства]
No (more) answers
[Ответов (больше) нет]

Если после этого рассматривать текст программы, то окажется, что отношение topic теперь имеет вид

topic (PTH conventional SMD),

а в отношении heading появилось новое утверждение

SMD heading (meaning methods advantages)

и, кроме того, к отношению data добавились три утверждения, каждое из которых характеризует новый элемент отношения heading.

6.3. ИСПОЛЬЗОВАНИЕ ЭКСПЕРТНЫХ СИСТЕМ ДЛЯ УПРАВЛЕНИЯ ПРОЦЕССАМИ

Хотелось бы надеяться, что примеры предыдущего раздела дадут читателям хотя бы смутное представление о том, какие операции используются в процессе производства печатных плат. На рис. 6.2 изображена схема, показывающая не только этапы производства печатных плат, но также и типы проверок, применяемых на каждом этапе. На самом деле схема должна быть более сложной и включать еще ряд этапов, но для иллюстрации основной идеи автором были выбраны только несколько операций. Важно, что эта схема является прототипом для широкого класса схем, описывающих любые строго последовательные процессы, в которых каждая операция может быть начата только после того, как все предшествующие этапы выполнены, и правильность их выполнения подтверждена. Что касается рис. 6.2, то на нем не нашел отражения ряд этапов: среди них этап подготовки к отправке, который включает хранение, упаковку и подготовку счетов, а также такой не всегда обязательный этап, как нанесение золотого покрытия. Ниже приведена программа 6.7, которая является, по существу, маршрутной картой работ по изготовлению печатных плат. Эта программа выполняет следующие функции:

- инициирует новые работы по указанию отдела сбыта;
- хранит последовательность выполненных операций и процедур контроля, проведенных для каждой отдельной работы;
- управляет ходом работ, разрешая переход к следующей стадии только в том случае, если все необходимые проверки выполнены;

готовит отчеты, содержащие данные о ходе выполнения данной работы или всех работ в целом.

Отношение new-job используется для того, чтобы дополнить список работ еще одной работой, и для того, чтобы включить предложение

X stage in

показывающее, что новая работа уже прошла через ряд операций предварительного этапа in. К этим операциям относятся объявление стандартов, которые должны соблюдаться, подготовка и проверка фотошаблонов, подготовка и проверка программ станков с ЧПУ, предназначенных для сверления отверстий, и, возможно, подготовка и тестирование прототипов. Для того чтобы все это реализовать на микроЭВМ, нужна программа, аналогичная только что приведенной; для большой ЭВМ отдельной программы может и не понадобиться — только что описанные процедуры можно включить в текст выше приведенной программы. Для того чтобы иметь дело с короткой программой, допустим, что операции предварительного этапа in были выполнены, а заключительный процесс cut * просто не рассматривается. В этом случае удастся сконцентрировать внимание непосредственно на процессе управления производством печатных плат.

Программа 6.7

```
(X Y) match ((X Z) (Y x))
(X Y) match ((Z | x) (y | z)) if
  (X Y) match (x z)
new-job if
  (job y) delete and
  (job number X) is-told and
  APPEND (Y (X) Z) and
  (job) Z add and
  (X stage in) add and
  /
  job (1 2 3)
process (cut-blanks tooling-holes drill electroless primary-image
  electroplate strip-resist etch reflow solder-resist legend-print
  final-inspection QA-release out)
standard (in op1 op2 op3 op4 op5 op6 QA1 QA2 QA3 op7 op8 QA4
  QA5 out)
X awaits Y if
  process Z and
  X stage x and
  APPEND (y (x) Y) Z)
X awaits (cut-blanks Y) if
  X stage in and
  /
```

* К этому процессу относятся процедуры, связанные с подготовкой к отпавке (хранение, упаковка и т. д.). — Прим. ред.

X next Y if
(job X operation Y inspection Z) is-told and
X awaits (Y|x) and
standard y and
process z and
(Y Z) match (x y) and
(Z stage X1) delete and
(X stage Y) add and
/
1 stage cut-blanks
2 stage electroplate
3 stage tooling-holes

Отношение standard содержит список стандартных тестов, соответствующих каждой стадии производства. На некоторых стадиях можно использовать несколько тестов; например, стадия травления требует проверки первого образца, проверок со стороны оператора и проверок по обеспечению качества. Проверка первого образца означает, что одна плата или небольшое число плат подвергается обработке только для того, чтобы путем проверок определить, являются ли изменения, внесенные в механизм данного процесса обработки и в используемые материалы, необходимыми. Эта предосторожность позволяет устранить дорогостоящие ошибки при серийном производстве плат. Под операторной проверкой подразумевается визуальный контроль некоторых или всех плат, который позволяет обнаружить и устранить очевидные дефекты того или иного процесса — например, травления или нанесения покрытия. Наиболее жесткий тип проверки — это обеспечение качества (QA). Этот тип включает операции, которые в некоторых случаях могут даже разрушать образцы. К ним относятся проверка глубины покрытия, сцепления соединений с подложкой, толщины покрытия отверстий и многое другое. Кроме того, обеспечение качества предусматривает использование различных установленных стандартов и может включать подготовку, кодирование и охранение информации о микроучастках платы, которую необходимо сделать доступной для контроля со стороны пользователей или института стандартов.

Традиционно различные этапы производства указываются в маршрутной карте, и по мере их прохождения оператор или инспектор по обеспечению качества, в случае если не все в порядке, делает пометку. Это дает возможность запретить переход к очередному этапу, если не все предыдущие этапы выполнены правильно. К несчастью, никто не может гарантировать, что все будет идти точно по плану. Платы, не удовлетворяющие стандартам, могут миновать некоторые этапы из-за невнимательности персонала. Но вероятность того, что дефектные платы покинут пределы завода, достаточно мала, поскольку они обязательно будут подвергнуты заключительному глобальному контролю. Тем не менее допущенные ошибки могут привести к задержкам выпуска продукции.

Перед тем, как разобраться, в чем может быть полезна программа, рассмотрим, как он работает.

Сначала определим последний этап, выполненный в ходе работы 1.

```
all (x: 1 stage x)
[определить все (x: 1 этап x)]
cut-blanks
[вырезание заготовок]
No (more) answers
[Ответов (больше) нет]
```

Полученный ответ свидетельствует о том, что этап, связанный с подготовкой заготовок для плат, выполнен, т. е. в результате из листа базового материала вырезаны заготовки нужного размера. Для определения процессов, которые необходимо выполнить для завершения работы 3, используем следующий запрос:

```
all (x: 3 awaits)
[определить все (x: 3 ожидает x)
(drill electroless primary-image ...)
[сверление химическая-активация-поверхности нанесение-рисунка-межсоединений-на-плату и т. д.]
No (more) answers
[Ответов (больше) нет]
```

Предположим теперь, что оператор хочет выполнить следующий этап работы 2. Анализ показывает, что последним в ходе работы 2 был этап, связанный с гальванопокрытием, а следующим по порядку является этап снятия резиста. Но для того, чтобы эта операция могла быть проведена, выполненная к этому моменту работа должна быть проверена с помощью контролирующей процедуры op6. То, что именно процедура op6 должна быть проведена перед выполнением этапа strip-resist, можно установить с помощью отношения next1, ставящего в соответствие последнему выполненному этапу элемент из списка standard. Для того чтобы перейти к выполнению очередного этапа, оператору необходимо правильно идентифицировать, во-первых, этот этап и, во-вторых, процедуру контроля, соответствующую этапу, который был выполнен последним. Следующий протокол демонстрирует две неудачные и одну удачную попытки продолжить работу 2:

```
all (x: 2 next x)
[определить все (x: 2 следующий этап x)]
job 2 operation X inspection Y ? ans etch op6
[работа 2 этап X проверка Y ? Ответ травление op6]
job 2 operation X inspection Y ? ans strip-resist op5
[работа 2 этап X проверка Y ? Ответ снятие-резиста op5]
job 2 operation X inspection Y ? ans strip-resist op6
[работа 2 этап X проверка Y ? Ответ снятие-резиста op6]
```

strip-resist
[снятие резиста]
No (more) answers
[Ответов (больше) нет]

Первая попытка привела к неудаче, поскольку платы не должны подвергаться травлению до тех пор, пока процедура снятия резиста не проведена и не подтверждена правильность ее выполнения. Вторая попытка неудачна из-за того, что в своем ответе о последней процедуре контроля оператор указал оп5, в то время как необходимо проверять правильность нанесения гальванического покрытия с помощью процедуры оп6.

На практике названия (или коды), содержащиеся в отношении standard, могут быть значительно более сложными. В этом случае они могут использоваться в качестве указателей к тестам, которые должны быть проведены на данном этапе. Этот вариант достаточно легко реализовать. Коды op1, QA1 и т. д. были сознательно упрощены для того, чтобы сделать работу программы понятной практически каждому. Важнее обеспечить программе возможность печатать коды на маршрутной карте или на магнитной полосе, прикрепленной к карте. Это позволит с помощью специальной аппаратуры анализировать их и предотвращать переход к очередному этапу до тех пор, пока все предыдущие этапы не прошли проверки. Такая аппаратура действует так же, как денежные автоматы в банках, или другие устройства подобного рода.

В целом экспертная система будет не только помогать следить за ходом каждой работы, но и контролировать ее, предотвращая переход к тем этапам, которые по тем или иным причинам выполнять преждевременно.

Отметим, что предназначенные для эксплуатации в промышленности системы должны быть значительно сложнее той системы, которая была только что описана. Дело в том, что в ней подробно рассматривался только один тип печатных плат — платы со сквозными металлизированными отверстиями; кроме того, многие этапы процесса производства таких плат были опущены. На самом же деле существует довольно много различных типов печатных плат и в производстве плат каждого типа используются характерные для этого типа дополнительные операции. Также для разных типов плат существуют разные стандарты, которые часто определяются условиями эксплуатации плат. Все это можно учесть, введя несколько предложений, каждое из которых описывает определенный процесс обработки. Кроме того, с каждым таким предложением необходимо связать предложения, содержащие информацию о стандартах. В принципе реальные системы, кроме масштабов, мало чем отличаются от описанной в этой книге. Правда, их можно реализовать только на более мощных ЭВМ, чем домашние.

В заключение покажем, как еще можно использовать экспертную систему:

all (x: new-job x)
[определить все (x: новая-работа x)]
job number x ? ans 43
[номер работы x ? ответ 43]
43
No (more) answers
[Ответов (больше) нет]

После этого предложение job будет иметь следующий вид:

(1 2 3 43)

и, кроме того, появится новое предложение

43 stage in

Теперь новая работа с номером 43 готова для выполнения этапа вырезания заготовок:

all (x y: x next y)
[определить все (x y: x следующий этап y)]
job X operation Y inspection Z ? ans 43 cut-blanks in
[работа X этап Y проверка Z ? ответ 43 вырезание-заготовок
предварительный контроль]
43 cut-blanks
[43 вырезание-заготовок]
No (more) answers
[Ответов (больше) нет]

Ответ 43 cut-blanks показывает, что работа 43 подготовлена для выполнения этапа вырезания заготовок. В более сложных системах, о которых было упомянуто выше, очередной этап будет реализован на машине только в случае, если ранее будет получен правильный код от оператора. Естественно, что работа машины, выполняющей операцию, должна быть согласована с работой устройства, которое, проверяя коды, принимает решение о допуске данной работы к операции. В нашем примере этот код должен представлять собой комбинацию номера работы и контрольного кода in. Таким образом, в общем случае с работой связывается код, соответствующий номеру работы и коду последней контрольной проверки.

Упражнение 6.5

Используйте текст программы 6.7 для определения:

- текущего этапа каждой работы;
- этапов, которые будут проведены для завершения работы 2;
- процедур контроля, необходимых для каждого этапа обработки;
- условий перехода работы 3 к следующему этапу.

Упражнение 6.6

Как уже было отмечено, программа 6.7 может быть взята за основу при построении систем, контролирующих ход последовательных процессов. Возьмите любой такой процесс и попробуйте построить для него аналогичную систему.

6.4. ИСПОЛЬЗОВАНИЕ ЭКСПЕРТНЫХ СИСТЕМ ПРИ ПРИНЯТИИ РЕШЕНИЙ

В настоящее время наиболее мощными являются, по-видимому, экспертные системы, которые используются для принятия решений в достаточно узких областях. В коммерческих и финансовых службах экспертные системы, как еще раньше компьютеры, нашли применение быстрее, чем в промышленной сфере. Сейчас экспертные системы используются в банковском деле, страховании, торговых операциях и в системах автоматического управления канцелярской деятельностью. Но главной областью их применения остается моделирование финансов; в этом случае можно создать модель рынка для некоторых товаров (или одного товара) и проверить ее функционирование, изменяя внешние факторы, такие, как спрос, конкуренция, уровень сельскохозяйственного и промышленного производства разных стран и т. п.

Эти системы могут играть важную роль в обучении представителей деловых кругов, так как любую ситуацию можно промоделировать и сопоставить решения, предложенные обучаемыми, с решениями, которые были либо в действительности приняты, либо получены экспертной системой, либо являются наиболее оптимальными в данных условиях. Такого рода системы можно также использовать для оценки торговых мероприятий компаний в случае небольших изменений ситуации в мире. Эффективность такого рода систем зависит, во-первых, от качества алгоритмов и правил, предложенных экспертом и, во-вторых, от способа, с помощью которого инженер знаний переходит от идей, высказанных экспертом, к их программной реализации. Наиболее мощными являются системы, которые обладают способностями к обучению, в чем-то аналогичными способностям человека. Следует отметить, что проектирование обучающих алгоритмов — довольно сложная проблема; дело в том, что поведение системы не должно сильно меняться при небольших изменениях внешних условий, но в то же самое время система должна быть достаточно чувствительной к тем небольшим изменениям, которые могут предвещать смену тенденций на рынке.

Прогнозы экспертов в основном базируются на знании того, что произошло в прошлом, и на способности интерпретировать исторические факты. Но иногда приносит пользу и интуиция экспертов, т. е. также знания, которые трудно выразить словами и тем более определять формально. Естественно, экспертные

системы не обладают интуицией, но проектировщик системы может, опираясь на те свои интуитивные догадки, которые принесли успех в прошлом, построить систему так, чтобы можно было изменить ее поведение в случае возникновения новых озарений. В большинстве случаев экспертная система не выносит окончательного решения, а дает только свои рекомендации тому человеку, который отвечает за принятые решения.

Ниже описывается экспертная система, предназначенная для предсказанных результатов матчей Английской футбольной лиги. Следует сразу сказать, что эта система не является непревзойденным оракулом — богатым сможет стать лишь тот, кто существенно улучшит ее возможности *. Но все же она вполне пригодна для демонстрации приемов передачи информации от эксперта к системе. Преимущество этой системы заключается в том, что ее работу легко проверить либо по текущим результатам, либо по результатам матчей предыдущих лет. Читатели, если пожелают, могут заменить выбранные автором правила своими собственными. Но запомните, что любое нововведение должно пройти проверку на большом числе матчей и его можно будет ввести в систему только в случае, если оно действительно улучшает работу системы. Те, кто все-таки пойдет по этому пути, вероятно, сконструируют свои, отличные от предложенной автором экспертные системы.

Сначала предположим, что удалось найти футбольного эксперта, способного предсказывать результаты матчей и согласившегося объяснить, как он это делает. Допустим, что эксперт начисляет команде очки на основании результатов проведенных ею игр. Правила начисления следующие.

1. Команде дается два очка, если у нее побед больше, чем игр, проигранных и сведенных вничью.

2. Команде дается одно очко, если у нее побед больше, чем поражений.

3. Команде дается два очка, если она не проиграла ни одной игры.

4. Команде дается одно очко, если она ни разу не сыграла вничью.

5. Команде дается одно очко, если она забила больше мячей, чем пропустила.

6. Команде дается два очка, если она забила мячей, по крайней мере, в 2 раза больше, чем пропустила.

7. Команде дается два очка, если число ее побед, по крайней мере, в 2 раза превосходит число поражений.

8. Команде гостей дается два дополнительных очка, если у нее побед больше, чем игр, проигранных и сведенных вничью.

Сразу отметим, что для каждой команды все матчи разбиваются на два класса: сыгранные на своем поле и на чужом. Приведенные

* Дело в том, что в Великобритании широко распространен футбольный тотализатор. — *Прим. ред.*

правила относятся только к матчам одного класса. Для команд-гостей и команд-хозяев должен быть вычислен средний голевой баланс. Для команд-хозяев средний голевой баланс определяется частным от деления числа забитых мячей на количество сыгранных дома матчей и числа пропущенных мячей — на число сыгранных дома матчей. Аналогично определяется голевой баланс команд-гостей. Результаты округляются до ближайшего целого. При формировании прогноза матча определяются два числа. Одно равняется сумме среднего числа мячей, забиваемых командой хозяев, и среднего числа мячей, пропускаемых командой гостей; второе равняется сумме среднего числа мячей, забиваемых командой гостей, и среднего числа мячей, пропускаемых командой хозяев. Первое добавляется к общему числу очков, полученных командой хозяев в результате применения правил 1—8, а второе добавляется к очкам, полученным командой гостей. В результате формируются два показателя: один — для хозяев, другой — для гостей, которые характеризуют их шансы во встрече друг с другом.

Приведенный выше перечень правил может показаться достаточно сложным и запутанным, но на самом деле эта информация довольно хорошо структурирована и в лучшую сторону отличается от данных, получаемых от экспертов в других областях знаний. Ниже приведена программная реализация только что описанной системы. В программе 6.8 не хранится информация о сыгранных матчах: эту информацию в обобщенном виде должен вводить показатель.

Программа 6.8

```

()tot 0
(X Y) tot Z if
    Y tot x and
    SUM (X x Z) and
    /
game if
    form KILL and
    (home form (X) and away form (Y)) is-told and
    (h form X) add and
    (a form Y) add and
    /
    X played Y if
        X form (Z x y | z) and
        SUM (y X1 Y) and
        /
    X ave (Y Z) if
        X form (x y z X1 Y1) and
        X played Z1 and
        TIMES (x1 Z1 X1) and
        SUM (x1 0.5 y1) and
        y1 INT Y and
        TIMES (z1 Z1 Y1) and
        SUM (z1 0.5 X2) and
        X2 INT Z and
        /

```

```

X score Y if
    h ave (Z x) and
    a ave (y z) and
    SUM (Z z X) and
    SUM (x y Z)
X plus 2 if
    X form (Y Z x | y) and
    SUM (Z x z) and
    x LESS Y
X plus 1 if
    X form (Y Z x | y) and
    x LESS Y
X plus 2 if
    X form (Y Z 0 | x)
X plus 1 if
    X form (Y 0 | Z)
X plus 1 if
    X form (Y Z x y z) and
    z LESS y
X plus 2 if
    X form (Y Z x y z) and
    TIMES (2 x X1) and
    X1 LESS y
X plus 2 if
    X form (Y Z x | y) and
    TIMES (2 x z) and
    z LESS Y
a plus 2 if
    a form (X Y Z | x) and
    SUM (Y Z y) and
    y LESS X

```

Восемь утверждений отношения plus соответствуют восьми правилам начисления очков. Отношение tot используется для определения общего числа очков, получаемых с помощью отношения plus; отношение played — для определения числа сыгранных матчей; ave — для определения среднего голевого баланса команды хозяев и команды гостей и, наконец, score — для предсказания, основанного на этих средних значениях счета матча. В отношении ave среднее значение округляется до ближайшего целого. Это делается с помощью добавления 0.5 (0,5) к полученному среднему и применения встроенного отношения INT, вычисляющего целую часть числа. Отношение game удаляет все существующие в данный момент утверждения отношения form и помещает в программу два новых утверждения в формате

```

h form (w d l f a)
a form (w d l f a)

```

Данные w, d, l, f, a для этих утверждений запрашиваются у пользователя; w, d, l — число побед, ничьих и поражений соответственно; f, a — число забитых и пропущенных мячей. Символ h

свидетельствует о том, что речь идет о команде хозяев, символ а — гостей. Таким образом, всегда программа работает с двумя утверждениями отношения `form`.

Для того чтобы завершить программу, необходимо добавить к ней еще одно отношение `result`. С его помощью как раз и обеспечивается формирование прогноза на матч.

```
X result (Y Z) if  
(X next) is-told and  
game and  
x isall (x: h plus x) and  
x tot y and  
z isall (z: a plus z) and  
z tot X1 and  
Y1 score Z1 and  
SUM (y Y1 Y) and  
SUM (X1 Z1 Z)
```

Отношение `result` вызывает `game` для того, чтобы ввести данные о командах, проводит все необходимые вычисления с помощью отношений `plus` и `score` и обеспечивает выдачу прогноза в виде двух целых чисел, характеризующих шансы команд. Условие `(X next)` дает возможность пользователям печатать либо названия команд, либо порядковый номер матча. Если же в ответ на запрос `X next ?` пользователь ответит `no`, то программа завершит работу. Данные о командах хозяев и гостей размещаются при вводе в двух списках, следующих один за другим. Приведенный ниже протокол дает представление о том, как используется программа:

```
all (x: result x)  
[определить все (x: результат-прогноза x)]  
X next ? ans 1  
[следующий ? ответ 1]  
home form (X) and away form (Y) ? ans (6 1 0 21 6) (2 2 3 9 11)  
[команда хозяев (X) и команда гостей (Y) ? ответ (6 1 0 21 6)  
(2 2 3 9 11)]  
(1 (15 2))  
X next ? no  
[X следующий ? нет]  
No (more) answers  
[Ответов (больше) нет]
```

Ответ `(1 (15 2))` свидетельствует о том, что в игре под номером 1 шансы хозяев оцениваются числом 15, а гостей — 2, т. е. экспертная система в данном случае предсказывает уверенную победу хозяев поля. Если вы хотите убрать из ответа скобки, используйте следующий запрос:

```
all (x y z: x result (y z))
```

В результате будет получен ответ: `1 15 2`.

Как уже указывалось выше, при работе с системой можно предусмотреть использование в ответе названий играющих команд:

```
all (x y: result y)
[определить все (x y: x результат-прогноза y)]
X next ? ans (Chelsea v Man Utd)
[X следующий ? ответ (Челси против Манчестер Юнайтед)]
home form (X) and away form (Y) ? ans (6 0 1 13 4) (6 1 0 17 5)
[Команда хозяев (X) и команда гостей (Y) ? Ответ (6 0 1 13 4)
(6 1 0 17 5)]
(Chelsea v Man Utd) (12 15)
[(Челси — Манчестер Юнайтед) (12 15)]
...
```

Ответ показывает, что небольшое преимущество отдается команде Манчестер-Юнайтед. И на самом деле счет в матче этих команд оказался 2 : 1 в пользу Манчестера.

Если никакой новой информации в программу не вводилось, отношение form будет хранить данные о командах Челси и Манчестер-Юнайтед. Используя эти данные, можно с помощью отношений plus, score и ave получить много полезной информации. Например, запрос

```
all (x y: x plus y)
```

позволяет получить список очков, назначаемых командам в соответствии с правилами 1—8. С помощью запроса

```
all (x: score x)
```

получаем основанный только на учете среднего голевого баланса прогноз — (3 3). Запрос

```
all (x: h ave x)
```

дает (2 1) — средний голевой баланс хозяев поля.

Запрос

```
all (x: a ave x)
```

в свою очередь тоже дает (2 1) — средний голевой баланс гостей.

Упражнение 6.7

Используйте только что описанную экспертную систему (программу 6.8) для предсказания результатов следующих матчей:

Арсенал (5 1 1 10 6) — Манчестер Сити (1 2 4 6 11)
Астон Вилла (1 3 3 9 10) — Оксфорд (0 3 5 10 23)
Куинз Парк (5 1 1 11 4) — Шеффилд Юнайтед (5 0 2 11 9)
Ипсвич (1 2 4 4 5) — Челси (1 3 3 6 11)

Приведенная в этом разделе программа, позволяющая прогнозировать результаты матчей, предоставляет широкие возможности

для экспериментирования. Конечно, никто не ожидает, что прогноз должен подтверждаться с вероятностью, близкой к 100%, но все же точность в 65—70% должна быть достижимой. Читатели могут попытаться усовершенствовать программу, внося изменения в отношение plus. Можно добавлять новые правила, удалять старые и изменять число добавляемых очков. Также имеется возможность более точно вычислять средний голевой баланс. Для этого нужно перед округлением умножить среднее на десять, тем самым сохраняя первую десятичную цифру. Такая модификация следующим образом изменит отношение ave.

```

X ave (Y Z) if
  X form (x y X1 Y1) and
  X played Z1 and
  TIMES (X1 Z1 X1) and
  TIMES (10 x1 y1) and
  SUM (y1 5.0E - 1 z1) and
  z1 INT Y and
  TIMES (X2 Z1 Y1) and
  TIMES (10 X2 Y2) and
  SUM (Y2 5.0E - 1 Z2) and
  Z2 INT Z and
/

```

В случае, если необходимо значительно повысить роль средних оценок, их можно перед округлением умножать не на десять, а на сто. Заслуживает также внимания использование показателей команд во всех играх, а не отдельно в играх на своем и чужом полях. В этом случае можно с помощью разных весовых коэффициентов все-таки разделить игры на своем и чужом полях.

Чемпионат Английской футбольной лиги был выбран в качестве прикладной области для нашего примера не случайно. Дело в том, что матчи этого чемпионата проводятся в течение всего сезона, и их результаты могут постоянно пополнять информационный запас системы.

Упражнение 6.8

Напомним, сославшись на материал по базам данных, что важно иметь возможность модифицировать любое правило, не прибегая при этом к слишком большому числу (а желательно и совсем без) переписываний других правил. В этом смысле два правила отношения plus являются неудачными. Найдите эти два правила и постарайтесь их улучшить.

6.5. СИСТЕМЫ, ОБЪЯСНЯЮЩИЕ ЛОГИКУ СВОЕЙ РАБОТЫ

Желательно, чтобы пользователь имел возможность узнать у системы, каким образом она приходит к найденному заключению или как она объясняет тот или иной шаг в цепочке вывода, приво-

дящий к этому заключению. Почему важно иметь такую возможность? Во-первых, если такая возможность есть, пользователь будет получать дополнительную информацию, которую он сможет использовать для принятия собственного решения, возможно, и не согласовывая его с выводами, сделанными экспертной системой. Во-вторых, это позволит точно указать причины, приводящие к принятию системой неправильных решений. И, наконец, в-третьих, это поможет пользователям использовать протоколы работы системы для углубления своих знаний. Внесение изменений в отношение result, а также введение новых отношений в предыдущую программу (6.8) дают возможность пользователям в случае необходимости получить краткое описание процесса вывода (см. программу 6.9), реализуемого системой.

Программа 6.9

```

X result Y if
  game and
  Z points x and
  y score z and
  SUM (Z y X) and
  SUM (x z Y) and
  ((PP home X Y away)) ? and
  choose
choose if
  (either explain or X result Y)
explain if
  (explain result) is-told and
  X points Y and
  Z score x and
  SUM (X Z y) and
  SUM (Y x z) and
  ((P The home side has X bonus points, the away side has Y points and I
    add these points to the scoring probabilities to obtain y z)) ?
X points Y if
  Z isall (Z: h plus Z) and
  x isall (x: a plus x) and
  Z tot X and
  x tot Y

```

Естественно, можно предусмотреть выдачу более детальной информации о процессе логического вывода. Так, в нашем примере можно в принципе проследить, как с помощью отношения plus производится начисление очков. Но автору кажется, что тратить время на объяснение слишком большого числа мелких деталей нецелесообразно. Перейдем к описанию новых отношений. Отношение points, используемое для подсчета общего числа очков, получаемого с помощью правил отношения plus, вводится, во-первых, для того, чтобы упростить отношение result, и, во-вторых, для того, чтобы обеспечить данными другое новое отношение — explain. Отношение choose осуществляет ту же функцию, что и оператор ветвления языка ассемблер или конструкция «if ...

then ... else» языка высокого уровня. По мнению автора, ветвление в этой версии Пролога реализовано достаточно просто и элегантно, по крайней мере не так, как в Бейсике, где для этой цели используются следующие операторы:

```
IF ... THEN GOTO ...  
IF ... THEN GOSUB ...
```

Отношение choose позволяет пользователю выбрать один из трех вариантов: либо получить объяснение поведения системы и продолжить работу, либо, не получив объяснения, перейти к анализу очередных данных, либо, наконец, ответив на оба вопроса системы по, выйти из системы. Если продолжить проведение аналогий с языками высокого уровня, то можно сказать, что подобные действия реализуются там с помощью операторов вызова процедур. Так, в данном случае result как бы вызывает chose, которое в свою очередь либо вызывает explain и затем result, либо вызывает только result, либо просто завершает работу. Достоинство Пролога заключается в том, что приведенная выше конструкция очень близка к предложению обычного естественного языка: «выберите либо объяснение поведения, либо переход к анализу новых данных, либо выход из системы». Следующий протокол демонстрирует работу усовершенствованной системы:

```
all (: result x)  
[определить все (: результат-прогноза x)]  
home form (X) and away form (Y) ? ans (8 0 0 21 4) (1 1 6 5 17)  
[команда хозяев (X) и команда гостей (Y) ? ответ (8 0 0 21 4)  
(1 1 6 5 17)]  
home 58 11 away  
[хозяева 58 11 гости]  
explain result ? yes  
[надо ли объяснять, как получен результат ? да]  
The home side has 11 bonus points, the away side has 0 points  
[команда хозяев имеет 11 премиальных очков, команда гостей —  
0]  
and I add these points to the scoring probabilities 47 and 11 to  
obtain (58 11)  
[к ним прибавляются очки, полученные в результате учета  
среднего голевого баланса, умноженного на 10, в результате  
получаем (58 11)]  
home form (X) and away form (Y) ? ans (4 1 2 14 9) (0 1 6 8 21)  
[команда хозяев (X) и команда гостей (Y) ? ответ (4 1 2 14 9)  
(0 1 6 8 21)]  
home 54 24 away  
[хозяева 54 24 гости]  
explain result ? no  
[надо ли объяснять, как получен результат ? нет]  
home form (X) and away form (Y) ? ans (1 2 4 4 9) (1 3 3 6 11)  
[команда хозяев (X) и команда гостей (Y) ? ответ (1 2 4 4 9)
```

```

(1 3 3 6 11)]
home 22 22 away
[хозяева 22 22 гости]
explain result ? по
[надо ли объяснять, как получен результат ? нет]
home form (X) and away form (Y) ? по
[команда хозяев (X) и команда гостей (Y) ? нет]
No (more) answers
[Ответов (больше) нет]

```

Отметим, что следует использовать запрос

```
all (: result x)
```

а не запрос

```
all (x: result x)
```

Упражнение 6.9

Попробуйте модифицировать отношение result таким образом, чтобы система считала победителем того, кто набирает не менее чем на десять очков больше, чем его соперник. В противном случае результатом прогноза должна являться ничья. Прогноз должен быть дан на естественном языке, и, кроме того, в программе должна быть сохранена возможность объяснить свое поведение.

6.6. СИСТЕМА, СПОСОБНАЯ К ОБУЧЕНИЮ

Как известно, способность к обучению является неотъемлемым свойством интеллекта. Эффективность экспертной системы можно значительно повысить, если предусмотреть в ней средства, контролирующие ее работу и в случае необходимости позволяющие изменить базу знаний. Эти средства можно сделать полностью автоматическими. Так, многие промышленные робототехнические системы осуществляют последовательности операций первый раз под управлением человека-оператора, а в дальнейшем управляют процессом сами. Аналогично, робот может быть снабжен сенсорными устройствами, позволяющими ему «чувствовать» внешнюю среду и реагировать на нее. Например, в простейшем случае робот, обладая способностями к осязанию, может находить объекты и определять их местоположение в некоторой области. Такой робот будет в состоянии также обнаружить появление новых и отсутствие старых объектов.

В тех областях, где экспертные системы используются либо для непосредственного принятия решений, либо как советчики, обратная связь, позволяющая определить, правильно ли было принято решение, должна быть установлена с помощью человека. Описанная выше система, предназначенная для прогнозирования результатов футбольных матчей, не будет знать о правильности своего предсказания до тех пор, пока мы не сообщим ей истинный результат. Если же такого сообщения нет, весьма возможно, что

система будет постоянно делать одни и те же ошибки, и тогда ее трудно будет назвать интеллектуальной. Для того чтобы избежать этого, можно выбрать один из двух путей. Остановимся сначала на первом пути. Предположим, что система в состоянии сообщать, какие факторы и как повлияли на ее решение. Тогда должны быть предусмотрены средства, позволяющие пользователю немного изменять правила для того, чтобы сделать менее вероятными неправильные прогнозы. Второй путь в принципе может дать больший эффект, но его труднее реализовать. Он заключается в наделении системы возможностями самой изменять собственные правила, исходя из результатов предыдущей работы. Начнем с первого, более легкого пути. Перед тем как использовать программу 6.10, удалите из программы 6.9 отношения result, choose и explain. Советуем Вам до этого скопировать программу 6.9 для того, чтобы сохранить ее для дальнейшей работы.

После удаления у Вас останется текст программы 6.8 с усовершенствованным вариантом отношения ave, в котором предусматривается перед округлением умножать средний голевой баланс на десять. Для того чтобы полностью сформировать программу, добавьте к этому тексту отношения, приведенные ниже.

Программа 6.10

```

X fcast (Y Z) if
  (next X) is-told and
  Y fcast1 Z
X fcast1 Y if
  (either Z result x and X plus Y or (edit which term y) is-told and
  y edit)
X result Y if
  game and
  Z points x and
  y score z and
  SUM (Z y X) and
  SUM (x z Y) and
  ((PP home X Y away)) ?
X points Y if
  Z isall (Z: h plus (x Z)) and
  y isall (y: a plus (z y)) and
  Z tot X and
  y tot Y
h plus (1 20) if
  h form (X Y Z | x) and
  SUM (Y Z y) and
  y LESS X
h plus (2 10) if
  h form (X Y Z | x) and
  Z LESS X
h plus (3 20) if
  h form (X Y 0 | Z)
h plus (4 10) if
  h form (X 0 Y)

```

```

h plus (5 10) if
  h form (X Y Z x y | z) and
  Z LESS X
h plus (6 20) if
  h form (X Y Z x y | z) and
  TIMES (2 y X1) and
  X1 LESS x
h plus (7 20) if
  h form (X Y Z | x) and
  TIMES (2 Z y) and
  y LESS X
a plus (8 40) if
  a form (X Y Z | x) and
  SUM (Y Z y) and
  y LESS X
a plus (9 10) if
  a form (X Y Z | x) and
  Z LESS X
a plus (10 20) if
  a FORM (X Y 0 | Z)
a plus (11 10) if
  A form (X 0 | Y)
a plus (12 10) if
  a form (X Y Z x y | z) and
  y LESS x
a plus (13 20) if
  a form (X Y Z x y | z) and
  TIMES (2 y X1) and
  X1 LESS x
a plus (14 20) if
  a form (X Y Z | x) and
  TIMES (2 Z y) and
  y LESS X

```

Полученная программа позволит Вам определить, какое именно правило отношения plus было использовано в процессе работы, и, кроме того, даст возможность в случае необходимости изменить это правило. Заметим, что отношения plus и points были модифицированы. Так, вторым аргументом всех утвержденных отношения plus является теперь двухэлементный список; его первый элемент служит идентификатором используемого правила, а второй элемент является вторым аргументом старого отношения plus, умноженным на десять. Семь утвержденных отношения plus используются для команд хозяев, столько же для гостей.

Отношение points преобразовано для того, чтобы привести его в соответствие с отношением plus.

Число очков, присуждаемых команде при использовании правил, было увеличено для того, чтобы расширить масштабы поправок, которые будут заключаться в увеличении или уменьшении числа добавляемых очков в зависимости от результата прогноза. Отношение feast позволяет пользователю взять уже сыгранный матч, получить прогноз и изменить, принимая во внимание истин-

ный результат матча, те правила отношения plus, которые его не устраивают. Отметим, что пользователю предоставляется возможность даже в случае неверного прогноза ничего не менять, ведь он может решить, что результат настолько неожидан, что никакая система в принципе не в состоянии его предсказать. В любом случае изменениям подвергаются только те правила, которые принимали участие в формировании прогноза. Было решено добавлять одно очко к каждому правилу, использованному в процессе выработки прогноза, который оказался верным, и отбирать очко при неудачном результате прогноза.

Читатели, если захотят, могут выработать свою процедуру поощрения и наказания. Модификация правила производится после ввода пользователем номера предложения в ответ на запрос системы. В данном случае модификация представляет собой довольно трудоемкую операцию, но кажется, что перед тем, как сделать процесс изменений полностью автоматическим, полезно все-таки еще раз проконтролировать работу системы:

```
all (x y z: x fcast (y z)
[определить все (x y z: x прогноз (y z))]
next X ? ans 1
[следующий X ? ответ 1]
home form (X) and away form (Y) ? ans (4 1 1 9 6) (1 0 5 3 12)
[команда хозяев (X) и команда гостей (Y) ответ (4 1 1 9 6)
(1 0 5 3 12)]
home 95 25 away
[хозяева 95 25 гости]
1 h (1 20)
1 h (2 10)
1 h (5 10)
1 h (7 20)
1 a (11 10)
edit which term X ? ans plus 1
[какое правило изменяется (X) ? ответ plus 1]
1 (h plus (1 20) if h form (X Y Z (x) и SUM (Y Z y) и SUM
(Y Z y) and y LESS X) *
[1 (h plus (1 20) if h form (X Y Z) x) и SUM (Y Z y) и y
LESS X)]
edit which term X ? ans plus 2
[какое правило изменяется X ? ответ plus 2]
...
edit which term X ? no
[какое правило изменяется X ? нет]
next X ? ans 2
[следующий X ? ответ 2]
home form (X) and away form (Y) ? ans (4 1 1 13 5) (0 1 5 7 17)
```

* После этого пользователь модифицирует указанное правило. — Прим. пер.

[команда хозяев (X) и команда гостей (Y) ? ответ (4 1 1 13 5)
 (0 1 5 7 17)]
 home 134 20 away
 [хозяева 134 20 гости]
 2 h (1 21)
 2 h (2 11)
 2 h (5 11)
 2 h (6 20)
 2 h (7 21)
 и т. д.

После выдачи сообщения

edit which term x ?

пользователь может выбрать для редактирования любое предложение и в зависимости от результатов прогноза либо увеличить, либо уменьшить число прибавляемых очков. После большего числа испытаний правила претерпевают ряд изменений и принимают вид, увеличивающий вероятность правильного прогноза. Коррекция правил — очень долгий и трудоемкий процесс и нецелесообразно его полностью перекладывать на пользователя. Рассмотрим, каким образом можно автоматизировать некоторые этапы этого процесса.

Поскольку на небольших ЭВМ возможности хранения данных ограничены, эффективность реализации на них такого рода систем невысока. Для таких систем необходимы компьютеры с жесткими дисками и хорошие программы управления файлами. Это позволит хранить данные о результатах матчей, проведенных в течение длительного времени. Тогда можно будет использовать программы, сопоставляющие правильность прогноза с истинным результатом и в случае необходимости вносящие изменения в свои собственные правила. Способность программ корректировать самих себя — очень мощное средство Пролога. Эффективно это средство можно использовать только на больших машинах.

Но вернемся к нашей системе. Процесс ее обучения можно регулировать автоматически. Приводимая ниже программа 6.11 осуществляет коррекцию своих собственных правил, опираясь на результат прогноза. Но пользователям все же необходимо вводить данные о командах и информировать систему о результате каждого прогноза. Для того чтобы упростить систему и сэкономить память, было решено не проводить вычисление среднего голевого баланса и, кроме того, сократить количество правил, предназначенных для начисления очков, с восьми до четырех: два — для команды хозяев и два — для команды гостей. Вместо среднего голевого баланса теперь подсчитывается, с одной стороны, сумма мячей, забитых хозяевами и пропущенных гостями, а с другой, — сумма мячей, пропущенных хозяевами и забитых гостями. Полученные числа участвуют в формировании показателей шансов команд во встрече друг с другом.

```

() tot 0
(X Y) tot Z if
    Y tot x and
    SUM (X x Z)
fcast if
    (home form X and away form Y) is-told and
    form KILL and
    p KILL and
    X result Y and
    (Right or wrong 1 /- 1 Z) is-told and
    x p y and
    x amend (y Z)
X result Y if
    (h form X) add and
    (a form Y) add and
    Z points x and
    (a p 0) add and
    ((PP home Z x away)) ? and
    /
X amend (Y Z) if
    (X factor (Y x)) delete and
    SUM (Z x y) and
    (X factor (Y y)) add and
    /
X points Y if
    Z isall (Z: h plus (x Z)) and
    y isall (y: a plus (z y)) and
    Z tot X1 and
    y tot Y1 and
    h form (Z1 x1 y1 z1 X2) and
    a form (Y2 Z2 x2 y2 z2) and
    SUM (X1 z1 X3) and
    SUM (Y1 y2 Y3) and
    SUM (X3 z2 X) and
    SUM (Y3 X2 Y) and
    /
X plus (1 Y) if
    Z factor (1 Y) and
    Z form (Z x y z) and
    y LESS Z and
    (X p 1) add
X plus (2 Y) if
    X factor (2 Y) and
    X form (Z x y z) and
    SUM (x y X1) and
    X1 LESS Z and
    (X p 2) add
h factor (1 10)
h factor (2 10)
a factor (1 10)
a factor (2 20)
a factor (0 0)

```

Отношение `fcst` принимает от пользователя данные об играющих командах и помещает их в базу данных, предварительно удалив все аналогичные данные. Затем оно используется для ввода оценки прогноза; оценка равна либо 1 (правильно), либо — 1 (неправильно). И, наконец, отношение `fcst` инициирует корректировку правил. Отношение `result` дает возможность оценить шансы команд, используя для этого отношения `points` и `plus`. `Points` определяет число очков, начисляемых каждой команде с помощью правил отношения `plus`. Отношение `plus` генерирует новое предложение вида

у р N

и включает его в программу. Здесь `у` принимает либо значение `h` для команды хозяев, либо `a` — для команды гостей; `N` — номер правила. Эта информация используется автоматическим механизмом корректировки, реализуемым с помощью отношения `amend`. Корректировка заключается в увеличении или уменьшении на единицу аргумента одного из правил отношения `factor`. Достоинство этого отношения заключается в том, что его присутствие позволяет использовать одно и то же предложение `plus` как для команд хозяев, так и для команд гостей. Заметим, что отношение `amend` использует только короткие предложения для удаления и добавления утверждений отношения `factor`. Это позволяет ускорить работу программы по сравнению с тем случаем, когда необходимо было корректировать все использованные правила отношения `plus`. Отметим, что утверждение

а р Ø

которое добавляется к программе при выполнении отношения `result`, является фиктивным. Оно необходимо для того, чтобы проверка условия

X р у

в отношении `fcst` не приводила к неудаче в тех случаях, когда ни одно из правил начисления очков не удалось применить.

Проанализируем предложение

а factor (Ø N)

где `N` — разность между числом правильных и неправильных прогнозов.

Заметим, что пользователи могут добавить свои собственные правила к отношению `plus`. Каждое дополнительное правило должно включать утверждение вида

(X р N) add

где `N` — порядковый номер дополнительного правила.

Вместе с дополнительными правилами должны также быть введены предложения вида

h factor (N M)

a factor (N M)

где M — число очков, начисляемых в случае успешного применения правила. Вводные пользователями правила будут так же, как и все остальные, корректироваться системой.

Ниже приведен диалог между пользователем и только что описанной экспертной системой

```
all (: fcast)
[определить все (: прогноз)]
home form X and away form Y ? ans (5 1 1 1 0 6) (1 0 6 3 13)
[команда хозяев X и команда гостей Y ? ответ (5 1 1 1 0 6)
(1 0 6 3 13)]
home 53 9 away
[хозяева 53 9 гости]
Right or wrong 1/ — 1 X ? just 1
[Верен или неверен прогноз 1/ — 1 X ? последний ответ 1]
(число пустых строк определяется количеством изменений, вно-
симых системой)
home form X and away form Y ? ans (6 0 0 15 0) (1 4 2 11 12)
[команда хозяев X и команда гостей Y ? ответ (6 0 0 15 0)
(1 4 2 11 12)]
home 59 11 away
[хозяева 59 11 гости]
Right or wrong 1/ — 1 X ? just —1
[Верен или не верен прогноз 1/ — 1 X ? последний ответ —1]
. . .
home form X and away form Y ? no
[команда хозяев X и команда гостей Y ? нет]
No more answers
[Ответов (больше) нет]
```

Ответ 1, оценивающий правильность первого прогноза, свидетельствует о том, что предсказание подтвердилось. В свою очередь, ответ —1 говорит о том, что прогноз, данный системой, неверен. В обоих случаях система применяет правила начисления очков только для команд хозяев. В связи с этим аргумент тех предложений отношения factor, которые относятся к хозяевам, сначала увеличивается на 1, а затем на нее же уменьшается, т. е. в конечном счете отношение factor не изменяется. Рассмотрим еще один пример работы системы:

```
all (: fact)
[определить все (: прогноз)]
home form X and away form Y ? ans (5 1 1 1 0 6) (1 2 5 6 12)
[команда хозяев X и команда гостей Y ? ответ (5 1 1 1 0 6)
(1 2 5 6 12)]
home 52 12 away
```

[хозяева 52 12 гости]

Right or wrong 1/ — 1 X ? just 1

[Верен или неверен прогноз 1/ — 1 X ? последний ответ 1]
home form X and away form Y ? ans (4 1 4 16 13) (4 2 2 11 12)

[Команда хозяев X и команда гостей Y ? Ответ (4 1 4 16 13)
(4 2 2 11 12)]

home 28 34 away

[Хозяева 23 34 гости]

Right or wrong 1/ — 1 X ? just 1

[Верен или неверен прогноз 1/ — 1 X ? последний ответ 1]
home form X and away form Y ? ans (4 1 4 17 11) (2 2 5 12 14)

[Команда хозяев X и команда гостей Y ? ответ (4 1 4 17 11)
(2 2 5 12 14)]

home 31 23 away

[Хозяева 31 23 гости]

Right or wrong 1/ — 1 X ? just 1

[Верен или неверен прогноз 1/ — 1 X ? последний ответ 1]
home form X and away form Y ? no

[Команда хозяев X и команда гостей Y ? нет]

No (more) answers

[Ответов (больше) нет]

В данном случае все три сделанные системой прогноза подтвердились. При формировании последнего прогноза ни одно из правил начисления очков не использовалось (прогноз был дан только на основании учета голевого баланса команд). Посмотрим, какой вид приняли предложения, входящие в состав отношения factor:

a factor (2 20)

h factor (2 21)

h factor (1 11)

a factor (1 11)

a factor (0 3)

Последнее предложение показывает, что все три прогноза оказались успешными. Аргументы предложений, относящихся к командам—хозяевам поля, были увеличены на 1, поскольку оба правила начисления очков были использованы по одному разу. На единицу был увеличен и аргумент одного предложения, относящегося к команде гостей, так как для гостей удалось применить только первое правило отношения rule. Приведем теперь пример неверного прогноза

all (: fcast)

[определить все (: прогноз)]

home form X and away form Y ? ans (4 2 3 15 8) (2 2 5 12 11)

[команда хозяев X и команда гостей Y ? ответ (4 2 3 15 8)
(2 2 5 12 11)]

home 37 20 away

[хозяева 37 20 гости]

Right or wrong 1/ — 1 ? just —1

[Верен или неверен прогноз 1/ — 1 ? последний ответ —1]
home form X and away form Y ? no

[команда хозяев X и команда гостей Y ? нет]

No (more) answers

[Ответов (больше) нет]

Предложения отношения factor теперь имеют следующий вид:

a factor (2 20)

h factor (2 21)

a factor (1 11)

h factor (1 10)

a factor (0 2)

Таким образом, можно сделать вывод, что применялось первое правило, и поскольку прогноз не подтвердился, аргумент соответствующего предложения factor был уменьшен на 1, т. е. вместо 11 он стал равным 10.

Хотя программа 6.11 невелика, она обладает довольно значительными возможностями и, что наиболее важно, позволяет проводить с ней многочисленные эксперименты. Область применения такого рода программ, естественно, не ограничивается предсказанием результатов футбольных матчей. Принципы, положенные в основу их создания, можно использовать в процессе разработки экспертных систем для любой области, в которой данные, накопленные к настоящему времени, обрабатываются с помощью множества правил. Эти правила первоначально определил эксперт, но они могут быть автоматически изменены после сопоставления реальных и полученных системой результатов. В качестве одной из таких областей может быть взята задача прогнозирования погоды. Пользователям домашних компьютеров в этом случае, как и во многих других, не стоит огорчаться из-за того, что на их машинах могут быть решены лишь небольшие задачи. Ведь качество используемых ими идей и степень общности, которой они в состоянии достичь, значительно важнее размера и сложности программной системы. Область, называемая искусственным интеллектом, без сомнения будет постоянно привлекать многих людей — и тех, кто просто получает удовольствие от решения интересных задач, и тех, кто профессионально занимается использованием вычислительной техники.

Ответы к упражнениям

Упражнение 6.1

- а) all (x: check x)
[определить все (x: проверка x)]
fault X ? ans no-play
[неисправность X ? ответ не-играет]
cassette-in ? y

```

[вставлена ли кассета? да]
  cassette-rewound ? y
[перемотана ли кассета? да]
  on-play ? y
[нажата ли клавиша «Пуск»? да]
  tape-moving ? no
[двигается ли лента? нет]
  (try motor fault)
[проверьте исправность электродвигателя]
  all (x: check x)
[определить все (x: проверка x)]
  fault X? ans motor
[неисправность X ? ответ электродвигатель]
  (12 v on motor ? y)
[напряжение на электродвигателе равно 12 В)? да]
  motor-running ? y
[электродвигатель вращается ? да]
  drivebelt ? y
[приводной ремень в порядке? да]
  idler ? y
[натяжной ролик в порядке? да]
  pinchwheel ? no
[маховик в порядке? нет]
  replace pinchwheel
[замените маховик]
б) all (x: check x)
  [определить все (x: проверка x)]
  fault X ? ? ans motor
  [неисправность X ? ответ электродвигатель]
  (12 v on motor ? y)
  [(напряжение на электродвигателе равно 12 В) ? да]
  и т. д.

```

Упражнения 6.2, 6.3

Ответы к этим упражнениям зависят от выбранной читателем предметной области, но следует отметить, что те методы, которые использовались в программах 6.1, 6.2 и 6.3, должны и здесь привести к успеху.

Упражнение 6.4

Используйте программу 6.5 для выбора в первом случае (а) метода сквозных металлизированных отверстий, а во втором случае (б) традиционного метода. После этого запрашивайте нужную информацию

Упражнение 6.5

- а) all (x : y x stage y)
- б) all (x: 2 awaits x)
- в) all (x y: process z and standard X and x ON z and y ON X and (x y) match (z X))
- г) all (x: 3 next x)
job 3 operation X inspection Y ? just drill op2

Упражнение 6.6

Все зависит от выбранного читателем последовательного процесса.

Упражнение 6.7

(9 2), (4 2), (11 12), (3 2)

А вот реальные результаты матчей: 1 : 0, 2 : 0, 1 : 1 и 0 : 2.

Упражнение 6.8

Неудачными являются пятое и шестое правила отношения plus. Может случиться так, что пользователи захотят ввести еще несколько элементов данных в список. Например, им может понадобиться s (количество) сыгранных матчей. Тогда список примет вид (w d l f a s). Чтобы указанные правила стали универсальными, необходимо использовать в них утверждения form вида

form (Y Z κ y z X!)

Утверждения такого вида позволяют работать с произвольным числом элементов в списке. Заметим, что при появлении новой информации, возможно придется добавить в программу еще несколько правил, предназначенных для начисления очков.

Упражнение 6.9

Одни из возможных методов предусматривает изменение отношения result и введение дополнительного отношения.

```
X result Y if
  game and
  Z points x and
  y score z and
  SUM (Z y X) and
  SUM (x z Y) and
  ((PP home X Y away))? and
  X fcast Y and
  choose
X fcast Y if
  SUM (X 9 Z) and
  Z LESS Y and
  ((PP This looks like an away win))? and
  /
X fcast Y if
  SUM (Y 9 Z) and
  Z LESS X and
  ((PP This should be a home win))? and
  /
X fcast Y if
  ((PP A possible draw))? and
  /
```


СПИСОК ЛИТЕРАТУРЫ

1. Clark, McCabe, Ennals, *Micro-Prolog Primer*, Sinclair 1983
2. McCabe, Clark, Brough, *Micro-Prolog Reference Manual*, Sinclair 1984
3. Weizenbaum, *Computer Power and Human Reason*, Pelican 1984
4. Rich, *Artificial Intelligence*, McGraw-Hill 1983
5. O'Shea (Ed.), *Artificial Intelligence*, Harper & Row 1984
6. Hayes and Michie (Ed.), *Intelligent Systems*, Ellis Horwood 1983
7. Gray, *Logic Algebra and Databases*, Ellis Horwood 1984
8. Aleksander, *Designing Intelligent Systems*, Kogan Page 1984
9. Lemmon, *Beginning Logic*, Nelson 1971
10. Bittinger, *Logic and Proof*, Addison-Wesley 1970
11. Burnham and Hall, *Prolog Programming and Applications*, MacMillan 1985
12. Gardner, *Further Mathematical Diversions*, Penguin 1969

ПРОИЗВОДСТВЕННОЕ ИЗДАНИЕ

Дж. Макаллистер

**ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ
И ПРОЛОГ НА МИКРОЭВМ**

Редактор **Д. П. Бут**
Художественный редактор **С. Н. Голубев**
Оформление художника **С. Н. Голубева**
Технический редактор **Н. М. Харитонова**
Корректор **Л. Я. Шабашова**

ИБ № 6576

Сдано в набор 02.02.90. Подписано в печать 24.05.90.
Формат 60×90^{1/16}. Бумага офсетная № 2. Гарнитура литературная.
Печать офсетная. Усл. печ. л. 15,0. Усл. кр.-отт. 15,0.
Уч.-изд. л. 12,80. Тираж 30 000 экз. Заказ 31. Цена 2 р. 40 к.

Ордена Трудового Красного Знамени издательство «Машиностроение»,
107076, Москва, Стромынский пер., 4

Типография № 6 ордена Трудового Красного Знамени
издательства «Машиностроение»
при Государственном комитете СССР по печати
193144, Ленинград, ул. Моисеевко, 10.

**ГОТОВЯТСЯ К ПЕЧАТИ
В ИЗДАТЕЛЬСТВЕ «МАШИНОСТРОЕНИЕ»:**

Искусственный интеллект: Применение в интегрированных производственных системах/Под ред. Э. Кусяка; Перевод с английского А. П. Фомина; Под ред. А. И. Дашенко, Е. В. Левнера. — М.: Машиностроение, 1991.

Книга открывает новую серию «Искусственный интеллект в промышленности», выпускаемую издательствами ИФС (Великобритания) и «Шпрингер» (ФРГ). В книге обобщен опыт применения методов искусственного интеллекта для решения сложных задач, возникающих при создании гибких и интегрированных производственных систем. Это первая в СССР переводная книга, рассматривающая проблему комплексно: от тактильных сенсоров и распознавания речи до управления автоматическими складами и баз данных.

Книга особенно рекомендуется инженерам, разрабатывающим промышленные роботы.

Рот К. Алгоритмизация конструирования с помощью каталогов/Перевод с немецкого В. И. Борзенко, К. В. Казарновского-Кроля, А. Л. Колосова; Под ред. Б. А. Березовского. — М.: Машиностроение, 1991.

Перевод книги автора из ФРГ является первым в нашей стране изложением систематизированного подхода к конструированию широкого спектра изделий машиностроения с помощью таблиц конструкторских решений (каталогов). При этом подходе задача раскладывается на элементарные подзадачи, известные варианты решения которых содержатся в каталогах. Наряду с большим числом готовых каталогов книга содержит указания по составлению новых и дополнению имеющихся каталогов. Изложенная концепция создает фундамент для автоматизации проектирования любого класса машин и механизмов.

Круг читателей: творчески работающие теоретики и практики конструирования, специалисты по теории машин и механизмов, методологи и разработчики машиностроительных САПР, а также студенты и аспиранты соответствующих специальностей.

Уважаемые читатели!

*Заказывайте книги издательства «Машиностроение»
в магазинах и отделах «Техническая книга»*

